МОНИТОР

журнал для программистов — практиков

4.92

ISSN 0869—3889

PRO ET CONTRA

стр. 31

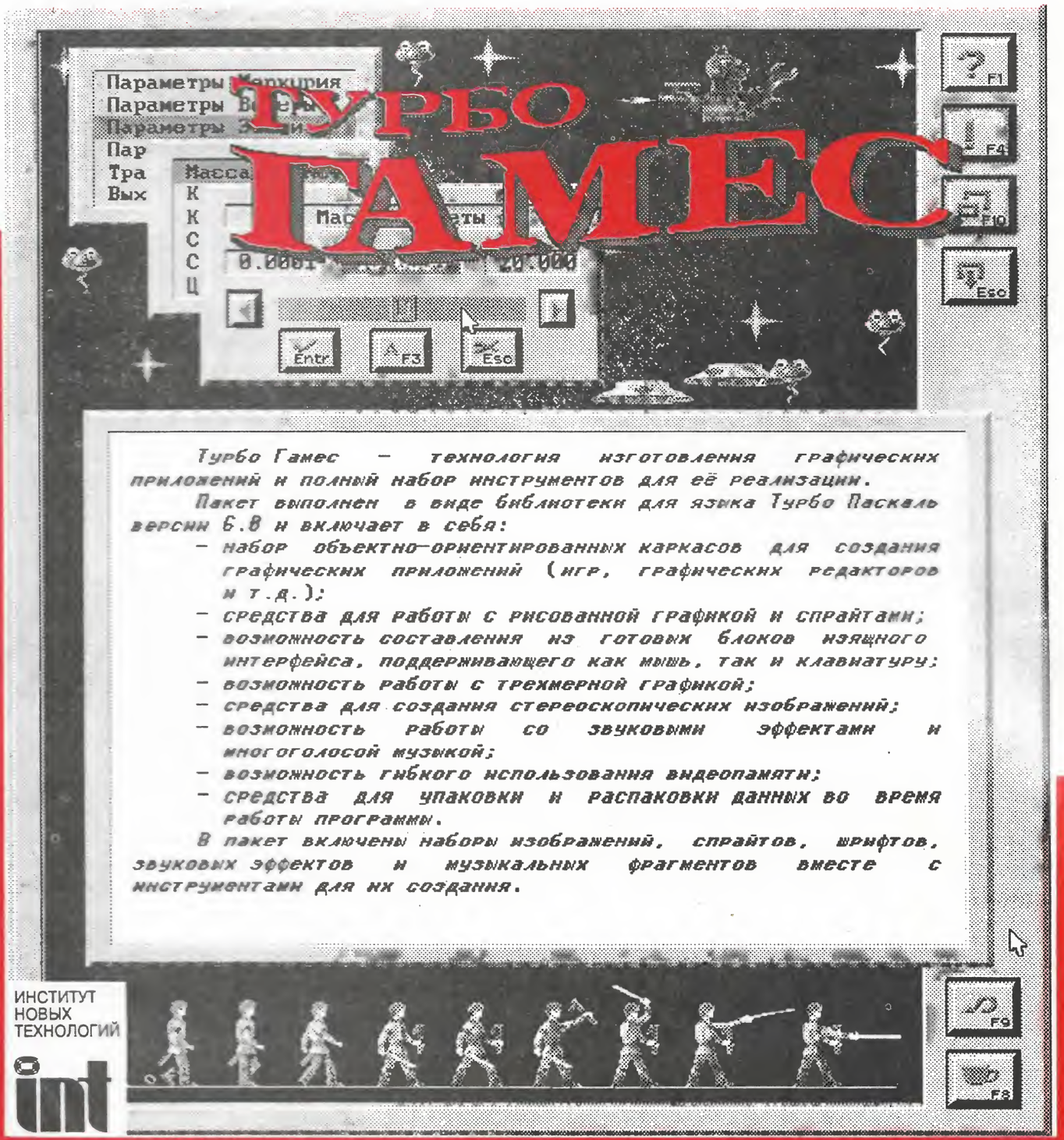
PROGRAMMER

CASE—технологии
Turbo Pascal 6.0
в ассортименте
От AutoCAD к CLDATA
и обратно

USER

Лемминги идут на север,
транспьютеры—на восток

Бесконечны комбинации модулей ТУРБО ГАМЕС, а создать игру Вы сможете за 1 день!



Турбо Гамес – технология изготовления графических приложений и полный набор инструментов для её реализации.

Пакет выполнен в виде библиотеки для языка Турбо Паскаль версии 6.0 и включает в себя:

- набор объектно-ориентированных каркасов для создания графических приложений (игр, графических редакторов и т.д.);
- средства для работы с рисованной графикой и спрайтами;
- возможность составления из готовых блоков изящного интерфейса, поддерживающего как мышь, так и клавиатуру;
- возможность работы с трехмерной графикой;
- средства для создания стереоскопических изображений;
- возможность работы со звуковыми эффектами и многоголосой музыкой;
- возможность гибкого использования видеопамяти;
- средства для упаковки и распаковки данных во время работы программы.

В пакет включены наборы изображений, спрайтов, шрифтов, звуковых эффектов и музыкальных фрагментов вместе с инструментами для их создания.

ИНСТИТУТ НОВЫХ ТЕХНОЛОГИЙ

int

Подробную информацию о ТУРБО ГАМЕС Вы можете получить по телефонам:

(095) 237-21-36,

(095) 247-36-25.

Куда ушли товарищи мои,
Что сердцу были дороги и милы?
Одни из них заплыли за буи,
А сил назад вернуться не хватило.

Другие в неположенных местах
Копали от рассвета до заката,
Они сложили головы в кустах,
Не вняв предупреждению плаката.

А третьих бес попутал, говорят.
Кого из нас не искушал лукавый?
Забыв про все, рванули в левый ряд,
Хоть был еще вполне свободен правый.

Четвертые - народец удалой,
Те никого на свете не боялись,
Стояли - руки в боки - под стрелой
И, по всему похоже, достоялись.

Ушли друзья в неведомую даль,
Наделав напоследок шуму много,
Но не скажу, что мне их очень жаль,
Ушли и ладно - скатертью дорога.

Я на своем остался берегу,
Решать свои конкретные задачи.
Я здесь стою и не могу иначе,
Но за других ручаться не могу.

Игорь Иртеньев

4 ноября прошлого года американский журнал "Business Week" опубликовал большую статью под заголовком "The Soviet Brain Drain" - "Утечка советских мозгов". Текст сопровождался улыбающимися фотографиями героев статьи - работающих в США советских (теперь - российских) ученых и инженеров: академика Роальда Сагдеева, создателя знаменитой компьютерной игры "Тетрис" Алексея Пажитнова, биолога - специалиста по СПИДу Евгения Шпайера, а также коллективной фотографией американской программистской фирмы, в которой 7 из 11 работающих - эмигранты из России.

Приведем несколько коротких цитат.

"В возрасте 36 лет академик Роальд Сагдеев стал самым молодым членом Академии Наук СССР. Пятью годами позже он возглавил престижный Институт космических исследований в Москве... В 1990 году он женился на внучке бывшего президента США Сьюзен Эйзенхауэр... Сейчас 58-летний Сагдеев - профессор физики в Мэрилендском Университете, где он проводит исследования по теории хаоса. "Советская наука, - жалуется он - является чем-то вроде исчезающего вида".

"В Беркли, Калифорния, Крис Донер, президент Access Softek, нанял на работу троих советских эмигрантов для разработки программ, работающих совместно с Microsoft Windows... Донер хочет нанять как можно больше русских в Москве, где "талантливые люди стоят пару сотен долларов в месяц".

"В проектировании программ для видеоигр трудно соперничать с 36-летним Алексеем Пажитновым, создателем "Тетриса"... Пажитнов придумал идею игры, работая над искусственным интеллектом в ВЦ АН СССР в Москве. Сейчас он работает в Bullet-Proof Software в Редмонде, штат Вашингтон, по трехлетней рабочей визе с возможностью остаться еще на три года... Основатель и директор Bullet-Proof Хэнк Роджерс... нанял 66 русских в Москве по бросовым ценам и говорит, что он извлекает большую выгоду из их интеллектуального подхода к развлечениям".

Пожалуй, хватит... Поскольку мы, в отличие от авторов "Business Week", живем и работаем там, откуда "утекают мозги", вряд ли мы разделим их восторг по поводу расцвета американской науки и промышленности за счет притока эмигрантов из России. Не станем мы восхищаться и тем, что та или иная группа российских программистов заключила "выгодный" контракт с западной фирмой, продав свои таланты и знания по цене "в 20 раз ниже рыночной". Почему-то не вызывает ожидавшихся эмоций и известие о "русификации" очередной западной программной системы.

Поскольку мы, в отличие от авторов "Business Week", живем здесь и сейчас, мы знаем, что гораздо опаснее внешней "утечки мозгов" "утечка мозгов" внутренняя. Вчерашние программисты продают сахар, гоняют из одного конца страны в другой вагоны с цементом, заседают в правлениях бирж и акционерных обществ. При всей разности занятий этих людей роднит одно - из сообщества программистов-профессионалов они выбыли, и, по-видимому, навсегда.

Каждый выбирает для себя... Поскольку мы, в отличие от авторов "Business Week", издаем журнал для программистов, а не для коммерсантов (относясь со всем уважением к представителям этой почтенной профессии), мы надеемся, что несмотря ни на какие "брэйн дрэйны" программистское дело в нашей стране ждут (и скоро!) лучшие времена. А на вопрос "Что делать?" ответ дает старая французская пословица: "C'est en forgean, qu'on devient forgeron" - "Чтобы стать кузнецом, надо ковать".

Журнал для программистов-практиков

Редколлегия:
Д.Э.Бройтман
(главный редактор)
Р.А.Болдырев
В.В.Водолазский
М.И.Гетманская
(отдел рекламы)
И.Г.Гун
С.В.Макельский
(главный художник)
А.М.Миляков
(ответственный секретарь)
А.В.Орехов
(издательский директор)
А.В.Свириденко
И.В.Сударева
Л.И.Устинова

В оформлении номера использованы рисунки
А.С.Замулы

Подписано в печать с оригинал-макета 20.07.92.
Формат 60x90/8.
Тираж 30 000 экз.
Печать офсетная. Печ.л.8.0.
Изд. N1971.
Цена договорная

Издатель:
Научно-техническое общество

Софт·Москва

Для писем: 117419, г.Москва,а/я 765,
тел. (095) 237-21-36, 247-36-25,
факс 126-31-56

© "Монитор", 1992

Технологии программирования

А.Штрик. **CASE: автоматизированное проектирование программного обеспечения** 4

Статья знакомит программиста с состоянием дел в области CASE - средств за рубежом.

Д.Арапов. **Объектно-ориентированный ввод-вывод в Turbo Pascal 6.0. Произвольный доступ** 7

В статье описаны библиотечные потоки в Turbo Pascal 6.0. Приводится критика существующей реализации.

Д.Солдатенков. **Единство кода и данных. Нетрадиционный подход к использованию объектно-ориентированных возможностей** 12

Первая статья из цикла статей о технологии создания компьютерных игр.

Н.Косарев. **Turbo Vision? Да!** 15

Данная статья не ставит перед собой задачу полного изложения принципов ООП. Однако некоторые практические моменты будут изложены ниже при описании приложения Turbo Vision, входящего в комплект поставки программного продукта Turbo Pascal 6.0 Professional.

В.Водолазкий. **Начальная установка обработчика событий в среде Turbo Vision** 21

Статья посвящена обработке событий в Turbo Vision (версия для языка Turbo Pascal 6.0).

Базы данных

Софткит™ выступает «ЗА» Clipper ®! 23

Интервью с участником 3-го Международного Компьютерного Форума.

А.Чернышков. **Clipper 5.0 - новые возможности Часть 4** 27

Несколько коротких программ, иллюстрирующих наиболее яркие возможности языка Clipper 5.0.

С.Крапивенко. **Clipper или Paradox: попытка аргументированного выбора Часть 1** 31

Автор статьи сопоставляет возможности этих систем, выделяет плюсы и минусы. Основное внимание уделено эффективности процесса разработки приложений. Даны описания некоторых важных команд и функций, приведены примеры программ. Знакомство с Paradox или Clipper желательно, но не обязательно.

Языки программирования

М.Черкашин. **Компилятор пишется так...** 37

Первая статья из цикла статей, посвященных проектированию компиляторов. Приводится текст простого компилятора на языке Pascal.

САПР	А.Крапивенко.	Программный интерфейс между системой AutoCAD и форматом CLDATA	44

Описывается формат двоичного файла обмена чертежами DXB и международный формат обмена чертежами CLDATA. Приводится текст программы - перекодировщика на языке С.

Параллельные вычисления	М.Дубицкий.	Транспьютеры. Введение в новую концепцию вычислительной техники	50

В статье приводятся теоретические модели и предпосылки транспьютерной технологии параллельных компьютеров. Первоначальное знакомство с транспьютерами и другими современными аппаратными средствами, выпускающимися в рамках транспьютерной технологии. Дается краткий обзор развития транспьютерного направления в России.

Защита программ	И.Василенко Е.Гулаков	Защищайтесь, сударь! Часть 1	53

Статья посвящена проблеме создания средств защиты от копирования для компьютеров IBM PC.

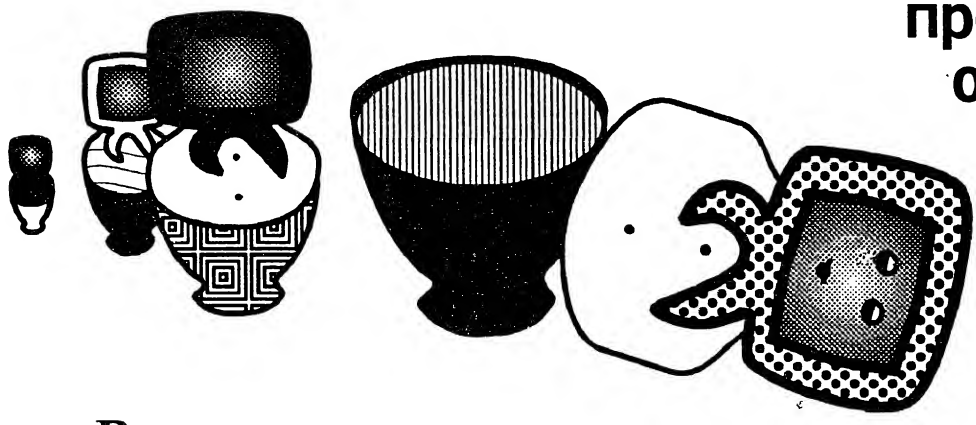
Игры	А.Лукашук.	Лемминги идут на север	59

Интересная информация счастливым обладателям игр LEMMINGS И LEMMINGS 2.
Управление, коды уровней, полезные советы.

Этюды для программистов	С.Курковский.	Письмо в редакцию	63

На досуге	А.Голубев.	Притчи о компьютерных жителях	64

CASE: автоматизированное проектирование программного обеспечения



А.Штрик

Концепции и основные положения

В происходящих в мировом сообществе процессах массовой информатизации особое место принадлежит проблемам создания программного обеспечения (ПО) для ЭВМ. Научно-методические и организационно-технические аспекты этих проблем рассматриваются прикладной дисциплиной Software Engineering (программная инженерия или технология программирования); необходимая же инструментальная поддержка сосредоточена в одном из разделов этой дисциплины - Computer Aided Software Engineering - CASE (автоматизированное проектирование программного обеспечения). CASE-технологии или CASE-средства согласно установившейся в зарубежных публикациях терминологии [1,2] - это совокупность регламентно-методических материалов, автоматизированных методов и инструментальных средств, применяемых для создания прикладного ПО как завершенного, пригодного к коммерческому распространению программного продукта, и гарантированно обеспечивающих заданные экономические показатели процесса создания ПО, а также показатели эффективности и качества конечного программного продукта.Таким образом, понятие CASE охватывает всевозможные методические и программно-инструментальные средства, объединенные общей целью эффективной поддержки любых процессов создания прикладного ПО.

Задача настоящей статьи - ознакомить программистов с зарубежными работами в области CASE-средств, включая эволюцию средств, общие принципы их построения, вопросы классификации, а также некоторые статистические данные по современным CASE-средствам, перечень и основные характеристики зарубежных CASE-средств, официально распространяемых в нашей стране.

Эволюция средств разработки программного обеспечения

При анализе эволюции средств разработки ПО выделяется ряд этапов, характеризующихся применением определенных видов инструментальных средств; примерами может служить применение ассемблеров, анализаторов и компиляторов, интерпретаторов, символических отладчиков и т.п. Для сегодняшнего дня характерны использование средств анализа требований, разработки спецификаций и проектирования структуры ПО (CASE-I) и реализация интегрированного окружения поддержки полного жизненного цикла (ЖЦ) разработки ПО (CASE-II). Два последних вида инструментальных средств, называемые за рубежом первым и вторым поколением (генерацией) CASE-систем, и составляют объект нашего последующего изучения.

Первое поколение CASE-I (1984-1990) было предназначено для системных аналитиков. Оно ориентировалось на персональные ЭВМ IBM PC под MS-DOS и на рабочие станции на базе DEC-UNIX и содержало средства сбора информации по требованиям заказчика, по предметной области и по проекту. Эта информация позволяла разработчикам ПО создавать диаграммы описаний системы и исходную документацию. По существу CASE-I явилась первой технологией, которая была адресована непосредственно системным аналитикам и предназначалась для роста производительности работ над проектом в целом, улучшения качества программ, а также для обеспечения контроля над процессом разработки. Этот подход был противоположен традиционным для 60-70-х гг. методам повышения производительности, которые были направлены на помощь программистам в процессах тестирования и отладки программ, проводимых после того, как в ПО при анализе и проектировании уже были заложены системные ошибки.

Главная цель CASE-I состояла в отделении процесса проектирования ПО от его кодирования, последующей разработки и производства. Поэтому и развитие этого поколения шло по линии создания методологий структурного проектирования ПО, разработки средств моделирования данных и структур баз данных, создания средств проектирования спецификаций ПО и других системных методов. Однако ограниченность функциональных возможностей и слабая интеграция со средствами разработки и производства программ стимулировали развитие CASE-средств этого поколения в сторону автоматизированной поддержки всего жизненного цикла ПО (ЖЦ ПО). Именно это и составляет цель второго поколения CASE-II; на коммерческом рынке оно появилось в 1990 году.

Все компоненты CASE-II могут быть разделены на ряд функциональных групп, каждая из которых решает определенные задачи в рамках ЖЦ ПО: стратегическое планирование проекта; моделирование информации предметной области; изучение возможностей решения проблемы; определение требований; системное проектирование; программирование; тестирование, отладка ПО и измерение качества; поддержка документирования; управление процессом проектирования; сопровождение. По имеющимся оценкам для реализации этих задач нужно свыше 100 различных инструментальных средств [3-5]. Пользователям предоставляется возможность выбора необходимого набора средств и их интеграции в нужном составе применительно к конкретным требованиям и условиям. Типовой состав CASE-II ориентирован на использование персональной ЭВМ совместно с универсальной ЭВМ. При этом на персональной ЭВМ размещаются все компоненты, характерные для CASE-I за исключением центральной базы данных проекта (repository), который заменяется небольшим локальным словарем. На универсальной ЭВМ размещаются центральный словарь и база данных проекта, средства генерации кода, средства управления проектом, словарь проекта, средства генерации отчетов и другие средства, используемые на различных этапах ЖЦ ПО.

Среди основных требований к современным CASE-средствам в первую очередь выделяется "человеческий фактор",

согласно которому эти средства обязаны сделать разработку ПО легким, удобным и экономичным процессом. Не менее важным требованием является широкое использование программных средств, уже получивших массовое распространение в других приложениях. К таким стандартным компонентам относятся в первую очередь базы данных, компиляторы, отладчики, документаторы, языки четвертого поколения 4GL и др. [3,4].

Классификация CASE-средств

Большое разнообразие существующих в настоящее время CASE-средств (более 500 наименований) требует наличия развитой системы их классификации. В повседневной практике используются в основном две: первая отражает функциональную ориентированность CASE-средств; вторая, более детальная, функционально-конструктивная, содержит развернутую совокупность различных показателей.

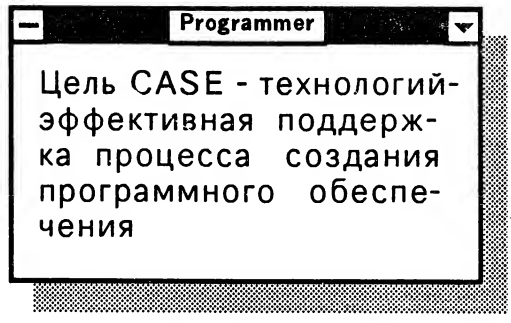
Первая классификация постоянно расширяется в соответствии с потребностями практики и конъюнктурой рынка. Тем не менее есть возможность выделить ряд устоявшихся в ней позиций, например:

- стратегическое планирование прикладной системы;
- управление и поддержку полного жизненного цикла ПО;
- управление качеством ПО;
- управление проектированием;
- анализ и проектирование ПО;
- генерацию кода;
- тестирование и отладку;
- интеграцию программной системы;
- управление конфигурацией ПО;
- словари данных и хранилища;
- создание прототипов;
- объектно-ориентированную обработку;
- повторную разработку ПО.

Функционально-конструктивная классификация CASE-средств отражает выполняемый перечень функций (функциональную наполненность) и структурную организацию (конструктивную реализацию); здесь CASE-средства делятся на категории, типы и виды.

Признак категория определяет степень интегрированности: вспомогательные (библиотечные) программы (tools); пакет разработчика, библиотеку программ (toolkit); инструментальное средство (workbench).

Категория tools обозначает вспомогательный пакет, решающий небольшую автономную задачу, принадлежащую процессу или проблеме более высокого масштаба. Однако иногда понятие tools используется также и в качестве обобщающего названия для всех категорий CASE-средств.



Категория toolkit представляет собой объединение интегрированных программных средств, которые концентрируются как правило на поддержке одного этапа разработки ПО, например, системного анализа, проектирования базы данных и т.д. Поскольку CASE toolkit специализируются на частичной автоматизации процесса разработки ПО, часто в одной разработке используются несколько различных toolkit.

Категория workbench по сравнению с toolkit обладает более высокой степенью интеграции выполняемых функций, большей самостоятельностью и автономностью использования, а также наличием тесной связи с системными и техническими средствами той аппаратно-вычислительной среды, на которой workbench функционирует. По существу workbench может рассматриваться как автоматизированное рабочее место или станция.

Признак тип обозначает функциональную ориентированность CASE-средств в технологическом процессе. Он позволяет группировать средства, имеющие общее функциональное назначение и применяемые в одних и тех же или близких стадиях ЖЦ, и потому позволяет быстро ориентироваться в весьма широкой совокупности имеющихся средств, отыскивать и сопоставлять аналогичные по назначению средства, выбирать подходящие к конкретным условиям. Как и во всей классификации CASE здесь нет постоянного перечня признаков классификации. В качестве таких признаков используются функции, поддерживаемые в технологическом процессе создания ПО и в частности приведенная классификация функциональной ориентированности.

Признак вид также связан с функциональной ориентированностью, однако в отличие от предыдущего определяет область действия в пределах ЖЦ: верхний (upper); средний (middle); нижний (lower).

Верхние CASE, иногда называемые системами компьютерного планирования, позволяют строить модель предметной области, отражающую механизмы функционирования, имеющиеся возможности, ресурсы, цели проекта.

Средние CASE - это средства поддержки этапов анализа требований и проектирования структуры ПО. Они анализируют информационные и конструктивные проблемы, используя диаграммы и словари данных, и существенно сокращают цикл разработки проекта. При этом немаловажную роль играет возможность накопления и хранения знаний, что позволяет использовать эти решения и через достаточно большое время при создании других проектов.

Нижние CASE - это средства, поддерживающие системы разработки ПО. Они используют компоненты для генерации программ и создания пользовательской документации. На этот уровень возложены также функции тестирования, управления конфигурацией, формирования полной документации, выдачи бумажных и экранных отчетов и т.д.

Опыт использования и характеристики современных зарубежных CASE-средств

Решающим критерием оценки эффективности использования CASE-средств большинство пользователей считает экономию времени разработки. К числу факторов, которые способствуют выполнению этого критерия, относятся расширение стандартизации; ускорение разработки; улучшение документированности; расширение возможностей создания прототипов; вовлечение пользователей в разработку; сокращение штатов; рост срока жизни ПО; улучшение контроля проекта. Вторым по частоте использования критерием эффективности является улучшение качества, третьим - экономия ресурсов [6,7].

Пять наиболее важных характеристик CASE-средств, на которые обращают внимание пользователи, включают: поддержку полного ЖЦ ПО и полноту интеграции; возможность создания прототипов; наличие центральной базы данных проекта; использование стандартных средств и методологий; открытость архитектуры CASE. Среди других характеристик пользователи выделяют поддержку разработки ПО для информационных систем и систем реального времени; использование различных типов систем управления базами данных; возможности управления проектом.

Статистическое исследование основных характеристик коммерчески доступных для зарубежных пользователей CASE-средств, проведенное автором настоящей статьи, позволило получить следующие данные.

1. Языки программирования, поддерживаемые CASE-средствами: отмеченное число языков - 22; наиболее распространены КОБОЛ, Си, АДА, ПЛ/1; количество языков, поддерживаемых одним CASE-средством, - до 7.

2. ЭВМ, на которых функционируют CASE-средства: отмеченное число типов ЭВМ - свыше 30; наиболее распространены IBM PC, DEC VAX, универсальные IBM Sun; количество ЭВМ, на которых функционирует один тип CASE-средств, - до 6.

3. Методы проектирования ПО, поддерживаемые CASE-средствами: отмеченное число методов - свыше 90; наиболее распространены методы Йодана (Yourdon), Де-Марко (DeMarco), Жане-Сарсона (Gane-Sarson), Чена (Chen); количество методов, поддерживаемых одним CASE-средством, - до 7.

4. "Популярность" CASE-средств. Общее число CASE-средств, имеющихся на рынке программных продуктов, около 500 наименований. Наиболее часто упоминаются такие CASE-средства, как CorVision, Excelerator/IS, Excelerator/RIS, Software Through Pictures, Procit*Workbench, Oracle и ряд других; общий перечень широкоизвестных средств включает несколько десятков. Отмечается прямая связь популяр-

Таблица. Официально распространяемые в России CASE-средства

СИСТЕМА, ФИРМА / ПОСТАВЩИК	НАЗНАЧЕНИЕ СИСТЕМЫ	ЯЗЫКИ ПРОГРАММИРОВАНИЯ	АППАРАТНАЯ ПЛАТФОРМА
Clipper5.0, Nantucket / СП Магнит	Система проектирования и программирования ПО любой сложности и объема; отличается открытой архитектурой и поддержкой объектно-ориентированного программирования	Clipper	IBM PC
CASE - ORACLE Oracle / ЛогоВАЗ -системы	Семейство программных CASE-продуктов автоматизации проектирования информационных систем; основана на нисходящем подходе к проектированию, позволяет последовательно отслеживать все этапы жизненного цикла ПО	SQL	IBM PC SUN Apple
Prokit Workbench, McDonnell Douglas / РосНИИ ИТ	Интегрированная система автоматизации проектирования ПО на стадиях стратегического планирования, проектирования и разработки информационных систем;основана на методологии STRADIS, ориентированной на поддержку концепции полного жизненного цикла информационных систем	PRO IY TRANSFORM TELON	IBM PC PS/2 IBM 3270-PC
OPEN ACCESS III, SOFTWARE PROD.INT./ СП Интерсофт	Семейство программных продуктов, предназначенных для автоматизации учрежденческой деятельности в различных прикладных сферах и для автоматизации научных исследований; может использоваться как инструментальное средство для создания прикладных систем и для самостоятельного решения прикладных задач	PROGRAMER с п/множ. SQL	IBM PC лок. сети
Семейство Design, Meta Soft.Corp. /Стиплер	Семейство инструментальных программ для создания ПО, использующее графический подход для представления и решения разнообразных проблем, а также визуализацию процесса мышления	Си	IBM PC PS/2 Compaq MacSE SUN
TopSpeed 3.0, Jensen & Partners Int./ Передовые технологии	Настраиваемая система программирования, включающая четыре компилятора, позволяет строить систему программирования, отвечающую конкретным требованиям на основе входящих средств TopSpeed Environment и TopSpeed TechKit. Все компиляторы используют общий оптимизирующий генератор кода	Modula Pascal C C++	IBM PC Windows OS/2
Object Vision, Borland	Инструментальная система для разработки прикладных задач в среде Windows, основана на принципе "наглядного программирования", позволяющем программисту - непрофессионалу создавать свои программы на основе логики работы проектируемой системы	Формы документов	IBM PC Windows

ности с тем, что такие средства обладают большим числом поддерживаемых языков программирования, методов проектирования и графических методов построения диаграмм.

Распространение зарубежных CASE-средств на российском рынке

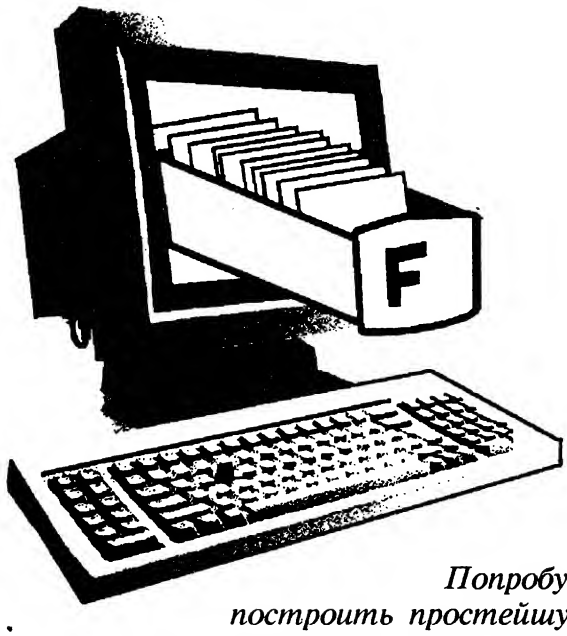
Перечень официально распространяемых в России CASE-средств различной степени интегрированности и их краткие характеристики приведены в таблице.

Перспективы развития

Одна из основных перспектив развития CASE-средств связана с третьим поколением, которое опирается в первую очередь на использование баз знаний и искусственного интеллекта. Кроме того активно ведутся работы по созданию мощного многопользовательского микрокомпьютерного применения CASE. Другие перспективы связаны с интеграцией CASE на полномасштабную поддержку ЖЦ с автоматическим получением машинных программ по свободно формируемым исходным требованиям. К числу перспективных относятся работы по стандартизации CASE, а также работы по созданию автоматизированных средств выбора и комплексирования конкретного набора CASE-средств в соответствии с требованиями пользователя на основе применения экспертных систем. ■

Литература

1. The scramble for CASE tools.- Software Magazine, March 1990, p.49-62.
2. Series on CASE reports. - American programmer, 1990,v.3, N 2,3,4.
3. E.Yourdon. The Texas instruments vision of CASE. - American programmer, April 1990, v. 3, pp. 1-7.
4. Oracle Corporation expands CASE Toolset.- Data base Product Reports, July 1989, v.7, N 8.
5. Grzanka L. Casing the Joint: How VAX Managers implement CASE.- Digital news, 1989, nov.27, p.35,36.
6. TOP 50.- Software Magazine, June 1991, p.23-64.
7. Hargraves R.F. How to plan CASE's Coming out Party for Your Organization.- Chief Information Officer Journal,



Объектно-ориентированный ввод-вывод в Turbo Pascal 6.0. Произвольный доступ

Д. Арапов

Попробуем построить простейшую объектно-ориентированную базу данных

Библиотечный метод доступа или TResourceFile

Нередко нам приходится работать с совокупностью объектов, которые не могут одновременно поместиться в оперативной памяти, и порядок обращения к ним заранее не известен. Напрашивается решение использовать произвольный доступ к потоку. Сразу заметим, что TStream содержит методы GetPos и Seek, обеспечивающие определение текущей позиции в потоке и его позиционирование. Однако из-за того, что длины объектов даже однотипных различны, в общем случае вычислить расположение интересующего нас объекта, не извлекая расположенных до него, невозможно.

Turbo Vision содержит объектный тип TResourceFile, названный по аналогии с соответствующими файлами в Windows, и представляющий из себя надстройку над потоками, реализующую библиотечный метод доступа. TResourceFile не наследует от TStream, а напротив является самостоятельной ветвью в иерархии объектов. TResourceFile содержит ссылку Stream на поток, в котором хранятся объекты. При этом, как и принято в объектно-ориентированном программировании, безразлично, с какого рода потоком он имеет дело: с TDosStream, TEmsStream или с каким-либо другим.

Имена методов TResourceFile напоминают имена методов потоков, но это совершенно другие по набору параметров и по сути методы. Методы TResourceFile.Get и TResourceFile.Put имеют на один аргумент больше, чем их тезки, определенные в TStream. При библиотечном методе доступа каждому хранимому объекту сопоставляется имя, в качестве которого может использоваться произвольная строка. Имя однозначно определяет, с каким объектом мы имеем дело.

Прежде чем детальнее ознакомиться с типом TResourceFile, рассмотрим его интерфейс:

```
type PResourceFile = ^TResourceFile;
   TResourceFile = object(TObject)
   tream      :  PStream;
   Modified :  Boolean;
   constructor Init(AStream: PStream);
   destructor  Done; virtual;
   function    Count: Integer;
   procedure   Delete(Key: String);
   procedure   Flush;
   function    Get(Key: String): PObject;
   function    KeyAt(I: Integer): String;
   procedure   Put(Item: PObject; Key: String);
   function    SwitchTo(AStream: PStream; Pack:Boolean):PStream;
end;
```

Ссылка на внутренний каталог скрыта от пользователя.

Перечислим основные методы объекта TResourceFile и выполняемые ими функции. Конструктор Init получает на входе ссылку на поток. Конкретный тип потока, как уже было сказано, значения не имеет. Деструктор Done выполняет необходимые действия по сохранению внутреннего каталога, а также закрывает и удаляет поток Stream.

Метод Put записывает объект в поток, а имя заносит во внутренний каталог. Функция Get достает объект по имени. Метод Flush сохраняет внутренний каталог (каталог - это наследник TCollection, у него имеются собственные методы Load и Store) в потоке. Определенная экономия достигается за счет поля Modified. Операции сохранения выполняют описанные выше действия только при значении true в этом поле.

TResourceFile содержит несколько вспомогательных методов. Во-первых, имеется возможность узнать число хранимых в нем объектов. Для этого служит функция Count. Во-вторых, с помощью обращения к функции KeyOf можно получить имя объекта по его порядковому номеру в каталоге (числу в диапазоне от 0 до Count-1), что в свою очередь позволяет обратиться к функции Get для получения самого объекта.

Из библиотечного потока можно удалить объект, для чего нужно сообщить его имя методу Delete. Новый объект записывается по заранее определенному имени, а прежде доступный исчезает. Следует заметить, что независимо от

того, удален ли объект с помощью процедуры Delete или в результате замещения другим объектом с тем же именем, удаляется только ссылка на него из каталога библиотеки. В потоке он по-прежнему будет занимать место. Если активно обновлять библиотечный поток, то он сильно "распухнет", при этом большую часть места в нем займут недоступные объекты. Для их удаления служит операция SwitchTo. Этот метод копирует объекты из потока, с которым работает TResourceFile, в новый. В зависимости от значения параметра Pack недоступные объекты можно либо копировать, либо нет. Окончив перенос объектов из потока в другой, SwitchTo закрывает исходный поток и переключается (отсюда и название метода) на его копию. Другая область применения SwitchTo - создание быстрой копии библиотечного потока, скажем, в дополнительно отображаемой памяти EMS.

Пример использования объектно-ориентированной базы данных

Рассмотрим пример построения на базе TResourceFile гипертекстовой обучающей системы. Предполагается наличие двух программ: авторской, с помощью которой наполняется база данных, и пользовательской, предназначенной для обучения. Возможности TResourceFile полностью раскрываются в авторской программе, но, к сожалению, она довольно велика, поэтому мы рассмотрим пользовательскую. При этом из нашего поля зрения выпадут операции Put,Delete и Flush, выполняемые при модификации набора данных. Система состоит из набора фреймов, появляющихся по одному на экране. Совокупность возможных типов фреймов описывается в модуле Frames. Для каждого типа фрейма там заданы методы Load и Store, а также регистрационные записи. Основным методом каждого фрейма является Execute. После его выполнения, во время которого и происходит взаимодействие программы с обучающимся, возвращается имя следующего фрейма. Мы предполагаем, что фрейм может быть размещен в доступной памяти целиком, в то время как объем всего набора значительно превышает возможности оперативной памяти. Абстрактный фрейм может быть описан следующим образом:

```
type PFrameBox =
     TFrameBox =
     object( TObject )
       function Execute : String; virtual;

     end;
```

Этот объект не содержит данных, и у него нет осмысленного конструктора. Реально в базе хранятся и выполняются наследники этого базисного объекта. Это могут быть гипертекстовые узлы, узлы, содержащие меню или карточки с вопросами, на которые требуется найти ответы и вписать в отведенное для этого место. Их все объединяет способность определять имя следующего фрейма и возвращать его в качестве результата метода Execute (пример).

Пример

```
program System;
uses Frames;
var AStream : PStream;          { Базовый поток          }
var DB      : PResourceFile;    { База данных            }
var Frame   : PFrameBox;        { Текущий фрейм          }
var Name    : String;           { Имя фрейма             }

begin
RegisterAllFrames; { Прежде чем работать с потоком, не    }
                   {обходимо зарегистрировать все типы.   }
    AStream:=new(PDosStream,Init('DBName.stm,stOpenRead));
                   { Открываем поток AStream на чтение.   }

DB := new(PResourceFile,Init(AStream)); { Открываем DB.   }
Name := DB^.KeyAt(0);{ Имена фреймов читающей системе     }
                     { неизвестны. Доступ к базе начинаем  }
                     { с первого фрейма.      }
while Name<>'' do begin { Пока есть куда переходить.      }
  Frame := TFrameBox(DB^.Get( Name ));{ Извлекаем фрейм   }
  Name := Current^.Execute; { Выполняем действия  и       }
                            { узнаем, в какой фрейм перейти }
                             { на следующем шаге.         }
  dispose( Frame,Done );{ Удаляем фрейм из памяти.        }
end; { while }
dispose( DB, Done ); { Закрываем базу данных.             }
end.
```

Читателя серьезно предупреждают

Рассмотрим подробнее, как работает TResourceFile. Этот объект содержит две ссылки: на поток (Stream) и на внутреннее оглавление. Последняя ссылка описана как приватная, клиенты TResourceFile не могут просматривать оглавление самостоятельно. Весь доступ к каталогу идет через методы. Отметим, что каждый элемент внутреннего каталога - это запись, состоящая из трех элементов: ссылки на имя объекта, смещения объекта относительно начала потока и длины образа объекта в потоке (рис. 1)

TResourceFile весьма удобен для хранения объектов, но ему присущи и некоторые недостатки, о которых следует знать прежде, чем решить использовать этот способ доступа.

1. В процессе работы внутренний каталог хранится в оперативной памяти, при этом каждый хранимый объект требует: 4 байта на хранящийся в коллекции указатель; 16 байтов займет один элемент каталога. Запись требует всего 12 байтов, но в куче могут быть выделены лишь участки, кратные 8 байтам. По той же причине как минимум 8 байтов требуется для имени объекта.

В итоге на один хранимый объект тратится 28 байтов оперативной памяти. Количество хранимых объектов ограничено максимальным числом элементов (16 380) в TCollection, на которой основан каталог.

2. Каталог записывается в конец потока. После того как выполнен хотя бы один Put в поток, образ нового объекта затирает хранимую копию каталога и в нашем распоряжении остается только актуальная копия в оперативной памяти (рис.

Programmer

Характерные применения объектно-ориентированных БД - картография,гипертекстовые системы, мультимедиа

2). Таким образом, любая чрезвычайная ситуация (сбой питания или ошибка в пользовательской программе), которая произошла между операциями Put и Flush, невосстановимо разрушает базу данных. Сгладить остроту проблемы могло бы соблюдение следующего правила: делать после каждой операции Put операцию Flush. В этом случае база разрушалась бы только при невозможности расширить поток (например, из-за нехватки места на диске) или ошибках в пользовательских методах Load и Store. Однако решиться принять это дисциплину нелегко: запись небольшого объекта (в несколько байтов) будет сопровождаться сохранением всего оглавления, а это может означать, как видно из сказанного ранее, запись до нескольких сотен килобайтов.

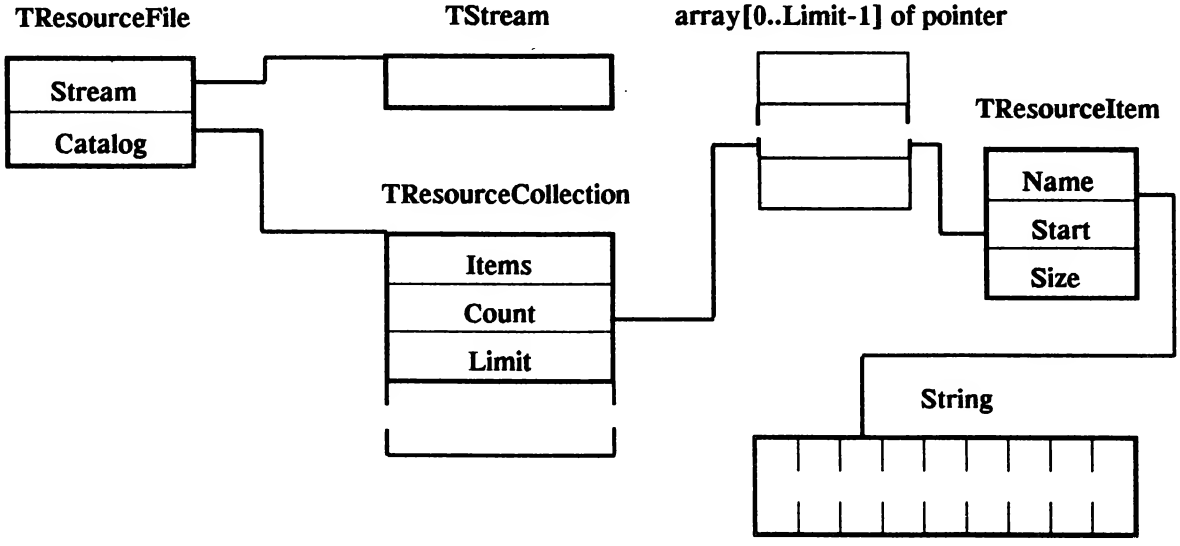


Рис. 1. Представление объекта TResourceFile в памяти

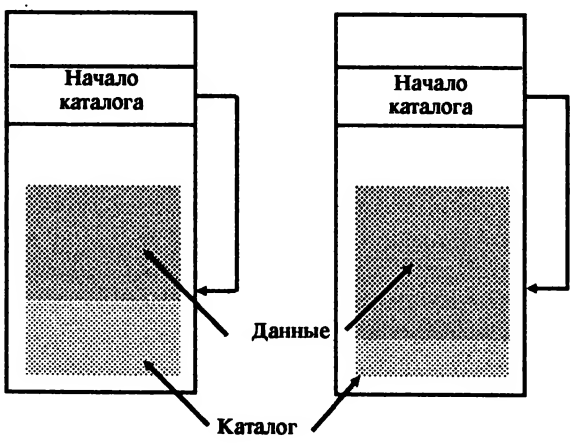


Рис. 2. Затирание каталога библиотечного потока

3. В процессе обновления библиотечный поток "распухает". (Доступ к более длинным потокам обычно требует и больше времени). Приходится периодически упаковывать поток, используя метод SwitchTo. У программиста, взявшего TResourceFile за основу интерактивной системы, имеется выбор из двух зол: вызывать упаковку по требованию пользователя, что по понятным причинам нежелательно, или по инициативе программ. В последнем случае система в неожиданные моменты оказывается на неопределенное время недоступной, что может пугать или раздражать пользователя.

На пути к объектно-ориентированным базам данных

Здесь мы попытаемся объяснить, как преодолеть недостатки, описанные в предыдущей части, и построить объект, который можно было бы назвать простейшей объектно-ориентированной базой данных.

Первый недостаток можно устранить, используя иерархические оглавления, имеющие, например структуру Б-дерева. При этом только часть оглавления присутствует в каждый момент в оперативной памяти. В данном случае достигается и то преимущество, что при внесении изменений в незначительное число объектов нужно записывать только измененные части оглавлений, при доступе к объекту также не обязательно считывать оглавление целиком. Этот прием позволяет не только сократить объем требуемой оперативной памяти, но и ускорить процесс открытия и закрытия базы. Другой прием еще проще. Во многих случаях нет нужды давать имена хранящимся в базе данных объектам, часто достаточно дать каждому уникальный номер, по которому на него будут ссылаться другие объекты. Таким образом, можно сэкономить как минимум 8 байтов, которые занимает в куче строка.

Недостатка, о котором идет речь во втором пункте, могло бы не быть, если бы поток поддерживал свою целостность по методу "Все или ничего". Для этого создадим еще один потомок потока TDosStream, добавив к Flush парный метод Discard. При вызове этого метода происходит откат к состоянию потока, бывшему на момент последнего выполнения Flush. Кроме того, метод Discard вызывается при ошибке вывода в поток. К сожалению, операционная сис-

тема не поддерживает подобного сервиса для файлов, но его можно реализовать с помощью разделения потока на страницы и использования таблицы страниц. Организация подобной структуры дело тонкое, поскольку таблица страниц должна обеспечивать рост потока, не занимать излишне много места, кроме того, по завершении транзакции переключение со старого комплекта страниц на новый должно выполняться аппаратурой по методу "Все или ничего".

Чтобы преодолеть трудности, описанные в третьем пункте, необходим "сборщик мусора", удаляющий недоступные объекты. Он должен вызываться системой в паузах. Заметим, что при интерактивной работе система почти все время находится в состоянии ожидания нажатия очередной клавиши. Достаточно разделить процесс сборки мусора на короткие, неделимые операции, после выполнения каждой из которых база данных вновь оказывается целостной и работоспособной, и мы получим сборщик мусора, который можно запускать во время периодов ожидания (скажем, по прерыванию DOS 28h, или из процедуры Idle при создании приложения в среде Turbo Vision). В качестве такой операции можно взять перенос одного объекта с места на место, однако это возможно лишь при определенных ограничениях на длину образа объекта в потоке. Если объект может иметь произвольный размер, то его перемещение возможно вызовет определенные трудности. Мы не знаем удачных способов решения этой проблемы для смеси объектов, длины которых существенно различны. Возможно следует искусственно разделять образы объектов на большие и малые, используя для малых описанную технику упаковки, а большие объекты хранить в другом потоке, выравнивая их по границам страниц и удаляя целыми страницами.

Объектно-ориентированные базы данных - бурно развивающаяся область программирования. Несколько таких баз (Orion, GemStone, ObjectStore, O2) уже доступны в среде операционной системы Unix. Наиболее наглядно преимущества таких баз видны при хранении разнородной, связанной между собой информации. При построении программных систем, работающих с данными многих типов, со структурированными данными и "навигационным" характером доступа, имеет смысл подумать о применении подобных систем. Характерные применения - хранение картографической информации, гипертекстовые системы и особенно мультимедиа. По сравнению с традиционными реляционными СУБД у них имеется один существенный недостаток - они плохо поддерживают массовые операции и поиск. ■

programmer

Кулинарные советы

1. Проверьте регистрацию объекта, включая каталоги.

2. После внесения изменения в библиотечный поток, не забудьте вызвать метод Flush.

3. Периодически упаковывайте библиотечные потоки.

4. Следите за наличием оперативной памяти для каталога. Если ее не хватит, библиотечный поток может погибнуть.

"ИНФОРМЭЙШН КОМПЬЮТЕР ЭНТЕРПРАЙЗ"
(РОССИЯ)
И "ИНТЕРНЕЙШЕНЕЛ ДЕЙТА ГРУП"
(США)

Приглашают Вас на 3–ю Международную выставку

"COMPUTERWORLD FORUM/MOSCOW",

проводимую 17–21 сентября 1992г. в пав. №4 Всероссийского Выставочного Центра

На выставке будут представлены новейшие ПК, периферийные устройства и прикладное программное обеспечение.

Организатор:

WORLD EXPO RUSSIA

129223 Москва, проспект Мира, ВВЦ, ПОК, 4 этаж, ком.409.
Телфон: 216-75-90,
Телефакс: 216-83-56.

Журнал "МОНИТОР"
приглашает к сотрудничеству

Частных лиц и организации, которые готовы взяться за распространение журнала в различных регионах России (особенно в Сибири и на Дальнем Востоке), Украины, Беларуси, в государствах Прибалтики, государствах Закавказья, в Казахстане, в Молдове, в государствах Средней Азии.

Возможно заключение соглашения о представительстве интересов журнала в Вашем городе или регионе (распространение, организация подписки, работа с рекламодателями, сбор информации) на взаимовыгодных условиях.

Тел.: (095) 237-21-36, (095) 247-36-25.
Факс.: (095) 126-31-56.
Адрес для писем: 117419, Москва, а/я 765.

Novintech

119517,Москва, ул.Нежинская,13,КОНЦЕРН "НОВЫЕ ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ" ("НОВИНТЕХ"),
МП "Системы и Комплексы". Телефоны (095) 441 11 00 (095) 442-57-92
Телефакс (095) 442 57 92

КОНЦЕРН "НОВЫЕ ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ" ("НОВИНТЕХ"), официальный дистрибьютер и представитель в СССР ведущих фирм мира в области ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ (Borland, Microsoft, Lotus, Symantec, Marstek, Nantucket и др.) МП "Системы и Комплексы" ПРЕДЛАГАЕТ:

(Цены указаны по состоянию на 1.04.92 г.,и могут быть изменены без уведомления)

ЯЗЫКИ ПРОГРАММИРОВАНИЯ

Borland C++ (вер.3.0) & Application Frameworks	24.300 руб
Borland C++ (вер.3.0)	16.020 руб
Turbo C++ for Windows	9.060 руб
Borland C++ Runtime (вер.2.0)	4.300 руб
Turbo C++ (вер.1.0 2nd Edition)	8.020 руб
Turbo Pascal (вер.6.0)	7.800 руб
Turbo Pascal Professional (вер.6.0)	11.400 руб
Turbo Pascal for Windows (вер.1.0)	10.560 руб
Turbo Pascal Runtime (вер.6.0)	5.800 руб
Turbo Pascal for Windows Runtime	5.800 руб
Turbo Debugger & Tools (вер.2.0)	7.940 руб
Turbo C++ & TurboVision	8.500 руб

БИБЛИОТЕКИ ФУНКЦИЙ

Object Professional для Turbo Pascal	34.350 руб
Turbo Professional для Turbo Pascal	21.750 руб
CGT, вер. 1.56 for Clipper 5.0	2.950 руб
Clipper Tools One for Clipper 5.0	3.765 руб

СУБД

Paradox (вер.3.5)	20.140 руб
Paradox Runtime (вер.3.5)	2.480 руб
Paradox LanPack (вер.3.5x)	6.540 руб
Paradox SQL Link (вер.1.1)	7.320 руб
Paradox SQL Link Rbd/VMS (вер.1.0)	8.040 руб
Paradox Engine (вер.2.0)	7.640 руб
Русификаторы для Paradox (версии 3.5x),	3.907 руб
Русификатор для Paradox Runtime	938 руб
Русификатор для Paradox LanPack	1.563 руб
Русификатор для Paradox Engine	2.344 руб
dBase IV (вер.1.5)	22.290 руб
dBase IV Lan Pack (вер.1.5)	27.800 руб
dBase IV Developer Edition	38.680 руб
dBase IV (русская вер.1.1)	14.950 руб
Clipper, вер.5.01 (Nantucket Corp.)	19.950 руб.

ЭЛЕКТРОННЫЕ ТАБЛИЦЫ

QuattroPro (вер.4.0)	8.520 руб
QuattroPro Lan Pack (вер.3.0)	7.080 руб
Framework III (русская версия)	16.320 руб

СКАНЕРЫ (с программным обеспечением)

Сканер (отечественного производства)	28.000 руб
MARS 105 Plus (Marstek Inc.)	42.650 руб

ИНСТРУМЕНТАЛЬНЫЕ СРЕДСТВА, САПР, РЕДАКТОРЫ, УТИЛИТЫ

ObjectVision (вер.2.0)	6.620 руб
ProtoGen (генерация C и C++ кода)	2.500 руб
SideKick (вер.2.0)	2.700 руб
Norton Commander	2.500 руб
Norton Backup	2.000 руб
Norton Utilities	3.000 руб
Norton Antivirus	2.000 руб
AutoCAD R10 (русская версия)	20.000 руб
AutoCAD R11/286 (русская версия)	32.000 руб
AutoCAD R11/386 (русская версия)	37.000 руб
SEMPRO - Семантический процессор	12.000 руб
MS WORD 5.0 + Русское Слово (вер. 2.0)	10.000 руб
Книга "Готовим тексты в MS WORD"	45 руб
Книга "Работа в Turbo C"	35 руб

Приобретая программные продукты в Концерне, Вы становитесь зарегистрированным пользователем всемирно известных фирм и получаете право на информационную и техническую поддержку, а также право на приобретение последующих версий программных продуктов со значительной скидкой.

Вы можете стать абонентом Системы Электронной Доски Объявлений по продукции фирмы BORLAND на базе АДС "МИР", которая работает круглосуточно.

МП "Системы и комплексы" Концерна "НОВЫЕ ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ" ("НОВИНТЕХ"), официальный дистрибьютер корпорации Borland International, предлагает:

dBASE IV v.1.5

- усовершенствованную версию языка dBASE, который является международным стандартом систем управления базами данных и создания приложений.

ВНИМАНИЕ! В течение ближайших 2-х месяцев dBASE IV v.1.5 продается со скидкой **50%** всем неофициальным пользователям dBASE, а также официальным пользователям системы Clipper (Nantucket Corp.).

Borland C++ 3.0 и Paradox Engine 2.0

- компилятор C и C++ кода для DOS и Windows вместе с библиотекой процедур и функций C и Pascal для создания баз данных и работы с всемирно известной базой данных Paradox

ВНИМАНИЕ! В течении ближайших двух месяцев при одновременной покупке этих двух продуктов, Вы покупаете их по цене UPGRADE!

10.420 + 4.340 = 14.760 вместо 23.660

Единство кода и данных. Нетрадиционный подход к использованию объектно-ориентированных возможностей

Д. Солдатенков

Можно ли работать с кодом точно так же, как с данными? Эта проблема возникла чуть ли не раньше самой вычислительной техники. Если программа может обращаться к своему собственному коду, то возникает возможность создания саморазвивающихся и саморазмножающихся программ, которая будоражит воображение. Именно с этой возможностью, обеспечиваемой MS-DOS, связан такой буйный расцвет компьютерных вирусов. Будем надеяться, что с переходом на систему Windows у создателей вирусов все-таки возникнут определенные трудности, связанные как раз с тем, что эта система не допускает такого вольного обращения с кодом, как DOS.

Та же идея - свалить в одну кучу данные и код, лежит в основе одного из принципов объектно-ориентированного программирования (ООП) - инкапсуляции. ООП стремительно вторглось в жизнь программистов и коренным образом ее изменило. Если раньше можно было смело писать, а потом думать, то теперь приходится делать все наоборот - сначала думать, а потом писать. Но этот небольшой недостаток с лихвой окупается теми многочисленными преимуществами, которые дает ООП.

Одним из лидеров ООП является Турбо Паскаль. Высокий уровень абстракции этого языка в сочетании с ООП позволяет создавать довольно изящные программы. Что касается данных и кода, то здесь каждому, кто хоть немного знает Паскаль, все ясно: данные описываются в виде переменных и констант, а код - в виде процедур и функций, и ничего общего между ними быть не может.

Но все дело в том, что процедуры и функции являются обычными типами. И мы запросто можем описать переменную или поле записи как Procedure, точно так же, как это делается с переменными типа Integer или Real. Иллюстрацией может служить программа VarFunc (пример 1).

Результатом работы этой программы будет выдача на экран двух разных сообщений, которые появляются в результате работы формально одной и той же процедуры, описанной как переменная.

С обычными процедурами и функциями все идет довольно гладко, но если использовать для подобных целей методы, внутренние процедуры объектов, то возникает ряд тонкостей.

Первая из них связана с тем, что методу передается больше переменных, чем обычной процедуре. Этой "лишней" переменной является указатель на объект, так называемый Self. Он заталкивается в стек первым и стоит последним в списке передаваемых переменных. Для того чтобы правильно вызвать метод как обычную процедуру, надо добавить в список передаваемых параметров этой процедуры еще один указатель.

Теперь формально переменная-процедура и метод принадлежат разным типам и прямого присваивания между ними компилятор сделать не даст, поэтому это присваивание надо провести на уровне указателей. В правой части присваивания должно стоять обязательно полное имя процедуры, иначе компилятор также не разрешит его (пример 2).

То, что необходимо указывать полное имя метода, исключает возможность использования виртуальных процедур. Попробуем устранить этот недостаток. Как известно, до виртуального метода можно добраться только посмотрев таблицу поздних связей после отработки конструктора. Таблица эта находится в сегменте данных, а смещение до нее находится сразу за последним полем данных в теле самого объекта. В таблице указате-

Пример 1

```
program VarFunc;
{$F+}
procedure Func1;
begin
    WriteLn ('Func1');
end;

procedure Func2;
begin
    WriteLn ('Func2');
end;

var
    Func: Procedure;

begin
    Func := Func1;
    Func;
    Func := Func2;
    Func;
end.
```

Пример 2

```
program VarMeth

type
    TObj1 = object
        constructor Init;
        destructor Done;
        procedure Meth1;
    end;

constructor TObj1.Init;
begin
end;

destructor TObj1.Done;
begin
end;

procedure TObj1.Meth1;
begin
    WriteLn ('Meth1');
end;

var
    Obj1: TObj1;
    Func: Procedure (Self: Pointer);

begin
    @Func := @TObj1.Meth1;
    Func (@Obj1); {Передаем методу указатель
        на обьект, которому метод принадлежит}
end.
```

ли на виртуальные методы располагаются в порядке их описания со смещения 4, по 4 байта на метод. Следующая программа VarVirtMeth обращается к таблице поздних связей и передает виртуальный метод в процедуру-переменную (пример 3).

Описанные выше приемы не являются бесполезными. На их основе уже написаны довольно крупные программы, которые осуществляют обработку совершенно разных по структуре и свойствам данных как однородого массива. При этом снимается ограничение на то, чтобы это были объекты с одним общим предком, и чтобы у этого предка все используемые методы уже были бы описаны.

Ситуация, где приведенный в программе VarVirMeth прием может дать некоторое преимущество над классическим подходом в объектно-ориентированном программировании, часто возникает при разработке программ компьютерных игр.

Если в игре участвует много объектов, использующих большое количество данных и незначительно отличающихся друг от друга (например только координатами), тогда невыгодно помещать эти данные внутрь каждого из объектов, равно как и делать их глобальными. В первом случае не эффективно используется память, во втором - появляется большое количество мусора, и мы прощаемся с одним из основных принципов ООП.

Один из путей решения этой проблемы заключается в том, что можно создать структуру только для той части данных, которая изменяется, зарезервировав некоторые из полей под методы. Таким образом, мы создаем как бы множество маленьких объектиков, которые можно будет обрабатывать как однородный массив. Основные же данные описываются как внутренние данные того объекта, из которого поставляются методы для маленьких объектиков-структур.

Примером может служить программа VarVirtMethApp (листинг).

При работе данной программы по черному экрану движутся разноцветные окружности, причем большие движутся слева направо, а маленькие - снизу вверх. Законы

Пример 3

```
program VarVirtMeth;

type
    TObj1 = object
        constructor Init;
        destructor Done;
        procedure Meth1; virtual;
    end;
    TObj2 = object (TObj1)
        procedure Meth1; virtual;
    end;
    PWord = ^Word;

const
    RecSz = 0;
    MethOff = 4;

constructor TObj1.Init;
begin
end;

destructor TObj1.Done;
begin
end;

procedure TObj1.Meth1;
begin
    WriteLn ('Meth1OfObj1');
end;

procedure TObj2.Meth1;
begin
    WriteLn ('Meth1OfObj2');
end;

var
    Dt: Byte;
    Obj1: TObj1;
    Obj2: TObj2;
    Func: Procedure (Self: Pointer);

begin
    Obj1.Init;
    Obj2.Init;
{ 1. Берем указатель на таблицу виртуальных методов    }
{ 2. Превращаем его в целое число                      }
{ 3. Добавляем смещение до нужного нам метода
     (номер метода * 4)                                }
{ 4. Снова превращаем в указатель                      }
{ 5. По этому адресу лежит указатель на искомую функцию}
{       5       4       2       1              3       }
{       T       T       T       T              T       }
{       v       v       v       v              v       }
@Func := Pointer (Pointer (LongInt (TypeOf (Obj1)) + MethOff)^);
Func (@Obj1);
Можно добраться до VMT без функции TipeOf
{ 1. Превращаем указатель на объект в целое число     }
{ 2. Добавляем смещение до VMT (RecSz = общий размер
     полей данных)                                     }
{ 3. По полученному адресу лежит слово                 }
{ 4. Добавляем смещение до нужного нам метода (номер
     метода * 4)                                       }
{ 5. Обращаемся в сегмент данных по этому смещению     }
{ 6. Там находится указатель на виртуальный метод      }
{       6       5       3       1       2       4      }
{       T       T       T       T       T       T      }
{       v       v       v       v       v       v      }
@Func := Pointer (Ptr (DSeg, PWord (LongInt (@Obj2) + RecSz)^ +
         + MethOff)^);
    Func (@Obj2);
    Obj1.Done;
    Obj2.Done;
end.
```

Листинг

```
program VarVirtMethApp;

uses Graph, Crt;

type
    PStare = ^TStare;
    TStare = record
        X, Y, Color: Integer;
        Self: Pointer;
        Move: Procedure (PSt: PStare; Self: Pointer);
    end;
        TObj1 = object
        constructor Init;
        procedure Meth1 (PSt: PStare); virtual;
        function PMeth1: Pointer;
        destructor Done;
    end;
    TObj2 = object (TObj1)
        procedure Meth1 (PSt: PStare); virtual;
    end;
    PWord = ^Word;

const
    GD: Integer = 3;
    GM: Integer = 1;
    RecSz = 0;
    MethOff = 4;

constructor TObj1.Init;
begin
end;

destructor TObj1.Done;
begin
end;

procedure TObj1.Meth1 (PSt: PStare);
begin
    with PSt^ do begin
        SetColor (0);
        Circle (X, Y, 20);
        Inc (X, 5); Inc (Y, 1);
        if X>640 then X:=0;
        if Y>350 then Y:=0;
        SetColor (Color);
        Circle (X, Y, 20);
    end;
end;

function TObj1.PMeth1: Pointer;
begin
PMeth1 := Pointer (Pointer (LongInt (TypeOf (Self)) +
        MethOff)^);
end;

procedure TObj2.Meth1 (PSt: PStare);
begin
    with PSt^ do begin
    SetColor (0);
    Circle (X, Y, 10);
    Inc (X, 1); Dec (Y, 5);
    if X>640 then X:=0;
    if Y<0 then Y:=350;
    SetColor (Color);
    Circle (X, Y, 10);
    end;
end;

var
    Stare: array [1..30] of TStare;
    Obj1: TObj1;
    Obj2: TObj2;
    i: Word;
    Ch: Char;
begin
    InitGraph (GD, GM, '');
    Obj1.Init;
    Obj2.Init;
    Randomize;
    for i:=1 to 30 do
        with Stare[i] do begin
            X := Random (640);
            Y := Random (350);
            Color := 1 + Random (15);
            if Color mod 2 = 0 then begin
                Self := @Obj1;
                @Move := Obj1.PMeth1;
            end
            else begin
                Self := @Obj2;
                @Move := Obj2.PMeth1;
            end;
        end;
    repeat
        for i:=1 to 30 do
            with Stare[i] do
            Move (@Stare[i], Self);
    until KeyPressed;
    Ch := ReadKey;
    Obj1.Done;
    Obj2.Done;
end.
```

■

движения окружностей описаны в виртуальных методах объектов двух типов - TObj1 и TObj2. Для экономии места сделано так, что эти объекты не имеют внутренних данных, но если бы они их имели, то эти данные были бы одни на все множество маленьких объектиков-структур.

Виртуальные методы передаются как поля в записи типа TStare, при этом как бы создаются самодельные объекты. Для передачи метода как параметра очень удобно использовать другой метод того же объекта. Функция PMeth1 возвращает указатель на процедуру Meth1, которая является виртуальной и может изменяться у дочерних объектов.

Вся дальнейшая обработка происходит внутри одного цикла, при этом между элементами массива не делается никаких различий. В такой массив можно собрать структуры разных типов и размеров, заполняя пробелами свободные места, и заставить выполняться в соответствующий момент совершенно произвольную процедуру или функцию. Уже существуют программы, работающие по этому принципу.

Конечно все вышеизложенные соображения применимы к C++.

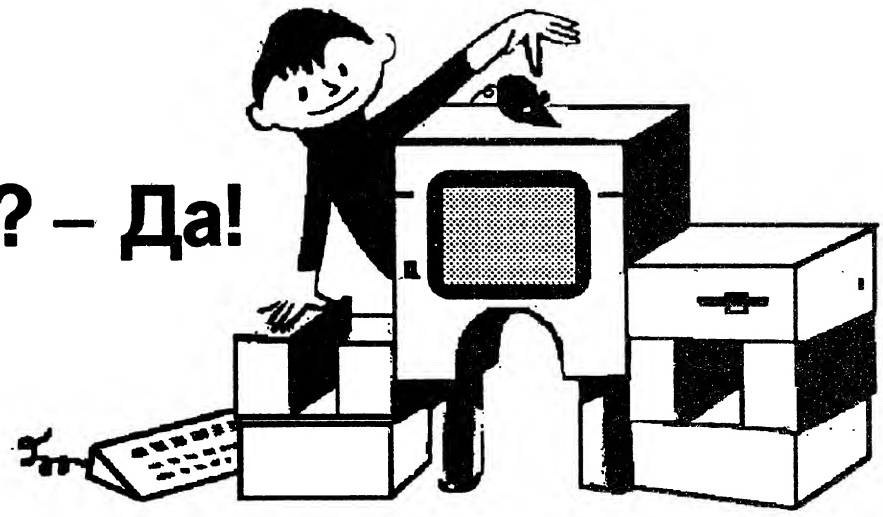
Эта заметка начинает цикл статей о технологии создания компьютерных игр. Далее речь пойдет о работе с рисованной графикой, создании спрайтов, трехмерной графике с уничтожением невидимых линий и отработкой освещенности, звуковых эффектах и многоголосой музыке. ■

Programmer

В некоторых случаях больший эффект принесет применение нетрадиционного подхода к использованию объектно - ориентированных возможностей. Этот подход широко используется при создании компьютерных игр

Turbo Vision? – Да!

Н. Косарев



Пролог, который можно не читать

" Стоит только недосмотреть за каким-нибудь незначительным явлением жизни, как оно сразу превращается в проблему и уже в качестве таковой начинает волновать умы."

И.Ильф, Е.Петров. "Пытка роскошью".

Изначально созданный Н.Виртом как учебный и отвечающий требованиям структурного программирования алгоритмический язык Pascal в виде различных коммерческих реализаций, самой популярной из которых является компилятор Turbo корпорации Borland, получил широчайшее распространение во всем мире. Немаловажное значение имеет тот факт, что устойчивому положению этого программного продукта на рынке инструментальных средств способствует грамотная коммерческая политика фирмы, сделавшей ставку на поддержку и развитие языков Pascal , C++ и отказавшейся от систем Turbo Basic и Turbo Prolog. Начиная с 1983 года, Pascal значительно изменился. Превращаясь из знаменитой 50-долларовой версии в Turbo Pascal 6.0, язык стал уделом не только любителей, но и профессионалов. В связи с этим в программистских кругах возникло мнение, что Pascal начинает утрачивать то, чем раньше привлекал многих : простоту, малые размеры, дешевизну.

Масла в этот разгорающийся скептический огонь добавили версии 5.5 и 6.0, включающие возможности объектно-ориентированной технологии разработки программ, и эта непонятная для непосвященных терминология: конструкторы, инкапсуляция и пр.

Апогеем подобных настроений явилась статья г-на Д.Джерролда "Еще один способ. Разрушители программ" в пер. Зверева В.В. (журнал БорИС, изд. БорАГ, вып. 4, октябрь 1991), в которой автор философствует о Turbo Pascal 6.0 и путях его развития, делает ряд критических замечаний относительно его возможностей, отмечает некоторые положительные черты. Суть его размышлений сводится к следующему.

- Наличие многооконного редактора, отладчика и компилятора в одном пакете - огромное достижение, но утрачена простота использования.

- Невозможность использования отладчика при компиляции программ свыше 80 K при наличии свободной памяти на машине автора 593 K. Автор проявляет недоумение по поводу неиспользования в Turbo Pascal 6.0 менеджера памяти VROOM.

- "Пропажа" FreePtr и частичная утрата совместимости Turbo Power Professional OOP версии 1.00 (OPRO).

Затем автор прошелся "по косточкам" Turbo Vision (TV) - объектно-ориентированного приложения к Turbo Pascal 6.0. Отметив, что это "мощное прикладное средство", с помощью которого создаваемые программы "будут выглядеть так, как если бы они поступили от "Borland", г-н Джерролд резюмирует, что TV - "смирительная рубашка" для программистов. Затем попутно сравнив два фрагмента текстов программ из TV и OPRO, автор достаточно субъективно высказывается по вопросу прозрачности синтаксиса и семантики в пользу последнего. Заканчивая статью, г-н Джерролд, предварительно сославшись на своего знакомого, своеобразно подводит итог, что поскольку Turbo Pascal на рынке находится в гордом одиночестве, то возможна близкая гибель языка.

Не вступая в полемику с автором этой статьи (в спорах, как известно, истина не рождается !), отмечу несколько моментов. По моему мнению, использование технологии VROOM M в Turbo Pascal явно вызовет повышение цены за этот продукт, а 593 K - это камень в огород Microsoft и кирпич - IBM. PC AT - и в Африке PC. 80 K - это разумный компромисс между быстродействием PC, памятью, используемой операционной системой и оболочкой IDE Turbo Pascal 6.0. Могу порекомендовать автору применить TPCX - компилятор Turbo Pascal, использующий EMS и входящий в комплект поставки Turbo Pascal 6.0 Professional.

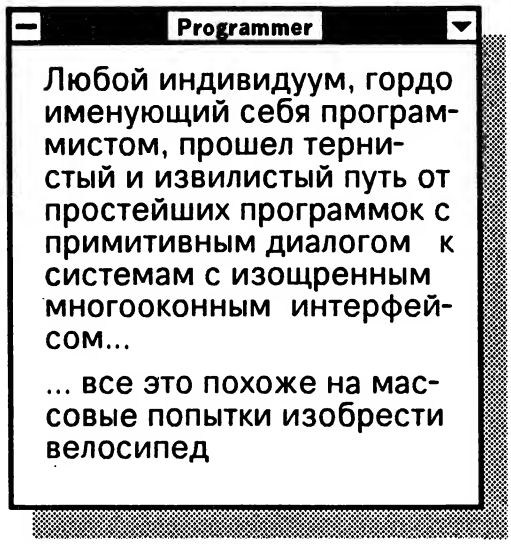
Что касается совместимости с пакетами, созданными ранее "под Turbo Pascal" - это прежде всего проблема не только этическая, но и программистская, и обсуждать ее, не имея достаточной информации о том, почему

фирма Borland сделала это, не имеет смысла.

TV - не панацея от всех бед программирования. Это не просто приложение к Turbo Pascal 6.0. Это - иллюстрация нового подхода к созданию программ.

Многие программисты, работающие в средах Turbo Pascal версий 5.5 и 6.0, отчасти из-за недостатка информации, а иногда и вовсе не принимая эти новшества, продолжают создавать программы в традиционном ключе и не используют преимуществ, которые предоставляет объектно-ориентированное программирование (ООП) - возможность разработки программ на более высоком уровне абстракции, с большей модульностью и структуризацией кода и данных.

Данная статья не ставит перед собой задачу полного изложения принципов ООП и их воплощения в Turbo Pascal. Однако некоторые практические моменты будут изложены ниже при описании приложения Turbo Vision, входящего в комплект поставки программного продукта Turbo Pascal 6.0 Professional.



Turbo Vision не догма, а руководство к действию

> "Хотя и нельзя еще было понять,
> что такое КЛООП, но по некоторым
> признакам замечалось, что учреждение
> любит новшества и здоровый прогресс".
>
> *И.Ильф, Е.Петров. "КЛООП".*

Любой индивидуум, гордо именующий себя программистом, прошел тернистый и извилистый путь ее величества эволюции от простейших программок с примитивным диалогом (через бейсиковский Input или паскалевский Readln) к системам с изощренным многооконным, многоцветным и многофункциональным пользовательским интерфейсом. Самоутверждение заставляет программистов разрабатывать свои собственные диалоговые средства. Не исключая в этих изысканиях большой творческой доли, стоит признать, что все это похоже на массовые попытки изобрести велосипед, а трата времени на разработку диалогового пакета для серьезной интерактивной системы - дорогое удовольствие.

Признанным стандартом графического "дружественного" интерфейса для PC является пресловутая оболочка Windows фирмы Microsoft. Взяв за образец и скопировав из Macintosh впечатляющую графику, с быстродействием у нее оказалось не столь благополучно. На PC модели AT 286 с экраном EGA работать с Windows просто утомительно. Если рассматривать весь спектр имеющихся в СНГ персональных компьютеров, то пользователей EC1840,41,42, "Искра 1030" и ее модификаций достаточно много. Адаптеры EGA для этих машин - пока экзотика. Windows для них - несбыточная мечта. Но эти компьютеры успешно решают задачи экономического характера на небольших предприятиях. Поэтому алфавитно - цифровой режим, реализуемый на IBM-подобных компьютерах, при умелом использовании может быть применен для реализации диалоговых систем на любой модели PC. Попутно отметим, что для создания собственных приложений "под Windows", чтобы использовать всю ее изобразительную мощь, необходимы дополнительно дорогостоящие и емкие инструментальные средства: например, SDK и Microsoft C, Borland C++, Pascal for Windows и непосредственно сама оболочка. Используя же Turbo Pascal 6.0 с объектно-ориентированным приложением Turbo Vision (и отказавшись от графики в диалоге), можно элегантно решить поставленную задачу.

Прежде всего, TV не является просто набором процедур и функций (типа Turbo Power Professional 5.05-5.09) и представляет собой разветвленную иерархию объектов. **Код программ TV отличается от обычного и полностью строится на технологии ООП, которая использует понятия объект, наследование, полиморфизм.**

Напомним, что под объектом (object) в Turbo Pascal понимают объединение полей данных и алгоритмов. Наследование представляет собой механизм переноса свойств от объекта-предка к объекту-потомку. Полиморфизм - разное выполнение некоторого метода с одним и тем же именем для объекта-предка и объекта-потомка.

Базовые объекты, представленные в TV, "сконструированы" таким образом, что с их помощью диалог в создаваемой программе приобретает Windows-подобный стиль (но, повторимся, в алфавитно- цифровом режиме !). **TV позволяет генерировать разные типы окон со скроллингом, обеспечивает автоматическую поддержку мыши, создание горизонтальных и вертикальных меню ,диалоговых окон ввода информации с "горячими клавишами" и знакомыми из Windows информационными боксами и функциональными кнопками "Ok" и "Cancel".**

(Вниманию эстетов - в TV имитируется нажатие кнопки). Есть возможность выполнения фоновых задач. В TV представлены также объекты, существенно расширяющие возможности языка Pascal. Это - так называемые коллекции, потоки, ресурсы.

Коллекция - это полиморфный динамический объект, который представляет собой динамический массив объектов и не объектов различных типов и размеров , содержит код итерационных методов для работы с элементами такого массива.

Потоки - это коллекции с определенным способом хранения кода и данных вне программы или в файле , или в расширенной памяти.

Ресурсы или точнее файлы ресурсов генерируются спе-

циальной программой - генератором, создаваемой пользователем и инициализирующей необходимые для основной программы объекты; последние записываются в ресурс или точнее в файл ресурса. Чтение объекта в основную программу из файла ресурса проще, чем инициализация, при этом EXE код программы становится на 10% меньше.

Уникален механизм создания контекстной помощи: при изменении размеров окна строка автоматически переносится на следующую, стремясь поместиться в окно с новыми размерами.

Заметим, что на средствах TV написана и сама интегрированная среда IDE Turbo Pascal 6.0.

TV содержит также исчерпывающий набор других элементов, не принадлежащих к иерархии базовых объектов. Здесь можно найти комбинации обычных клавиш с Alt, Control, Shift, F1-F10 и другие, аналог функции vsprintf (из C)- FormatStr и много других полезных переменных, функций и процедур.

Структурно TV состоит из набора модулей, осуществляющих полную системную поддержку создаваемой программы. Часть модулей, содержащих базовые типы объектов, не представлена в исходных текстах, а имеет TPU формат с описанием интерфейсных частей и включает следующие файлы : Objects, Views, TextView, Menus, HistList, Dialogs, Drivers, Memory. Это гораздо меньше, чем монстры типа OPRO. Вторая часть модулей, представленных в исходных текстах - App, Buffers, ColorSel, Editors, MsgBox, StdDlg - содержит типы объектов - потомков (от базовых) и используется в многочисленных примерах, приведенных в TV (пример механизма наследования !)

Для поддержки диалога - окон с элементами ввода данных, кнопок выбора вариантов, панелей выбора файлов, смены каталога и пр. предназначены модули StdDlg и Dialogs. При выводе сообщений и организации "стандартных" окон ввода полезны функции, содержащиеся в модуле MsgBox. Если проектируемая программа будет работать с текстами, то необходимые типы объектов можно найти в модулях Editors, Buffers, TextView. В модуль Memory включены процедуры манипуляции с хипом. Создание списков "истории" - просмотр и выборка ранее введеных строк информации - обеспечиваются объектами, описанными в HistList. Поддержка клавиатуры, мыши и монитора событий осуществляется драйверами модуля (Drivers). Проблему настройки цветовой гаммы в пользовательских программах можно осуществить, применяя объекты ColorSel. Это далеко не полный функциональный перечень модулей TV. Самое главное, что нужно помнить при работе с TV - это не разрозненный набор функций и процедур, а множество взаимосвязанных объектов.

Что, попробуем ?

- Что же я буду сотворять?
- Атомы с ядрами ! Орехи с зернышками !
Вишни с косточками !

Ж.Эффель. "Сотворение мира". т. 1.

Типичная программа "на Turbo Vision" выглядит следующим образом:

```
var AnyProgram : PAnyProgram;

begin
    PAnyProgram.Init;
    PAnyProgram.Run;
    PAnyProgram.Done;
end.
```

Расшифруем эту лаконичную запись. При вызове конструктора AnyProgram.Init происходят инициализация рабочей панели экрана, статус строки основного меню, а также вызов конструкторов, не видимых на экране, но использующихся в программе объектов. В AnyProgram.Run обрабатываются возникающие в программе события и выполнение функциональных блоков, созданных программистом . Дестрактор AnyProgram.Done уничтожает созданные видимые и невидимые объекты и освобождает память.

Для создания программы, которая что-то выполняет полезное, необходимо определить тип TAnyProgram:

```
type
    PAnyProgram : ^TAnyProgram;
    TAnyProgram : object(TApplication);
    {
        ...
        Процедуры инициализации меню и статус-строки,
        процедуры и функции пользователя,
        т.е. определяющие прикладные свойства объект-программы
        ...
    }
end;
```

Тип TApplication описан в модуле App.pas. Экземпляр AnyProgram, порожденный от предка TApplication, наследует обработчик событий, монитор экрана, менеджер памяти и систему обработки ошибок и не содержит прикладных методов.

Как правило, "сердцем" любой диалоговой программы является цикл с опросом кодов используемых клавиш. В TV программисту этого делать не потребуется. Базовые объекты TV сами обрабатывают значительную часть событий. Поэтому объекты, изображаемые на экране, "знают", как реагировать на манипуляции пользователя с клавиатурой и мышью. Дело в том, что каждый объект - потомок наследует метод HandleEvent, где это и происходит. Программист может внести изменения в поведение видимого элемента или дополнить его новыми свойствами. В общем случае поступают следующим образом:

```
procedure Потомок.HandleEvent(var Событие);
begin
    {
    ...Код, изменяющий поведение потомка в сравнении
     с предком - перехват события
    }

    Предок.HandleEvent(var Событие);

    {
    ...Код, добавляющий новые свойства в поведение
потомка
    }

end;
```

Для размещения объекта в динамической памяти, имеющего виртуальные методы, для версий Turbo Pascal 5.5 и 6.0 был расширен синтаксис оператора New:

New(тип-указатель_объекта, конструктор_объекта);

Например:

```
New(PPoint,Init(10,20);
New(PDialog,Init(R,'Dialog'));
```

Это позволяет компактно (одним оператором) инициа-

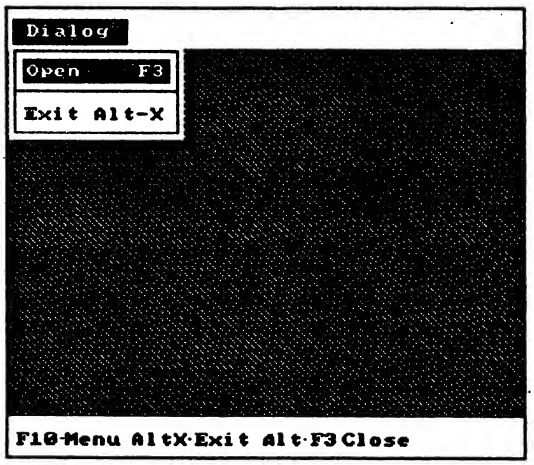


Рис.1

лизировать объект.

Рассмотрим программу, основные части которой взяты из примеров, представленных в TV, и прокомментируем основные моменты. Представленная ниже программа

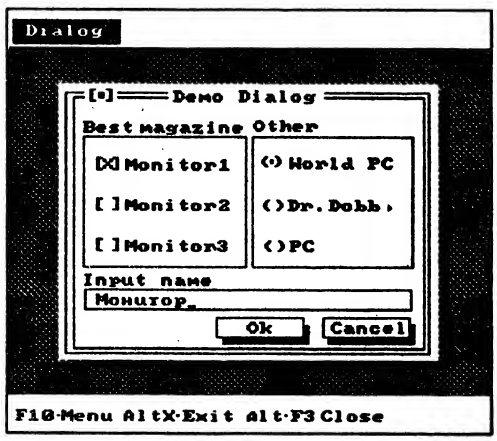


Рис.2

(пример) создает рабочую поверхность, статус-строку, меню с подменю (рис.1), из которого вызывается диалоговое окно "Demo Dialog". Оно содержит переключатели независимых (Best magazine) и зависимых кнопок (Other), строку ввода (Input name), функциональные кнопки Ok и Cancel, а также закрывающую кнопку [■] для мыши (рис.2).

Кроме этого, реализованы передача данных из диалога в программу и автоматическая поддержка мыши. На это требуется сто строк с небольшим!

Пример

```
{*                          *}
{*          with TV         *}
{*                          *}
program Demo;

uses Objects, Drivers, Views, Menus, Dialogs, App;

const
    cmDemoDialog = 101; { Команда, определяемая }
            { программистом, где                    }
    { ... } { cmXXX-мнемоника команды, принятая в TV.  }
            { По команде вызывается соответствующий метод}

type                    { Тип, представляющий структуру, поля }
    TData = record      { которой предназначены для хранения }
        CheckBoxData : Word; { вводимых в диалоге данных }
        RadioButtonData: Word;
        InputLineData  : string[128];
    end;

var
 Data : TData; { Переменная, в которой будут храниться данные }

type
    TMyApp = object(TApplication) { TMyApp - потомок }
                                  { объект-программы}
                                    { TApplication.     }
    procedure HandleEvent(var Event: TEvent); virtual;
                                    {Перекрывается,}
                                  { чтобы обработать команды }
                                  { программиста }
    procedure InitMenuBar; virtual; { Перекрывается, чтобы }
                          { создать "свое" меню}

    procedure InitStatusLine; virtual; { Перекрывается }
                                { для создания статус-строки}

    procedure DemoDialog;{Метод определяется программистом,}
                  { в данном случае, реализация диалогового}
 end;             { окна }

PDemoDialog = ^TDemoDialog; { Тип-указатель на объект}
                                {типа TDialog  }
                                 { необходим }
                                { для созданияи диалогового окна}
 end;

{ TMyApp }
{ Реализация обработчика событий }
procedure TMyApp.HandleEvent(var Event: TEvent);
begin
 TApplication.HandleEvent(Event); { Прямой вызов обработчика
                                  { событий }
                                  { TApplication, }
                                  { т.е. поведение TApplication }
                                  { не изменяется.}
```

Пример . Продолжение.

```
                                    { Теперь добавим реакцию }
                                    { на введенную команду: }
if Event.What = evCommand then { если происшедшее событие -}
                                                  { команда ... }
begin
    case Event.Command of
    cmDemoDialog: DemoDialog { и если команда = cmDemoDialog,}
                                        {  то выполнить  }
{ метод DemoDialog. }
        {...}           { Тут добавляется обработка других }
                        { команд, если }
                        { их доопределить и добавить }
                        { соответствующие им }
        else            { методы }
        Exit;
    end;

    ClearEvent(Event); { "Очистить" событие }
    end;
end;

{ Создание основного меню программы }

{ В констракторе Init(R,NewMenu(...) программист организует
вложенность подпунктов NewItem(...) в пунктах
NewSubMenu(...). NewSubMenu содержит ссылки на наименова-
ние, контекст подсказки (hcXXX) и указатель на первый пункт
подменю . NewItem - возвращает указатель на оба названия эле-
мента меню, код "горячей" клавиши (kbXXX), команду, выполня-
емую в этом пункте меню (в т.ч. определяемую пользователем ),
контекст подсказки и указатель на следующий пункт или nil, ес-
ли дальше пунктов нет. NewLine - указатель на пустую строку в
меню }

procedure TMyApp.InitMenuBar;
var R: TRect;
begin
    GetExtent(R);       { Установка координат }
    R.B.Y := R.A.Y + 1;
    MenuBar := New(PMenuBar, Init(R, NewMenu( { ~ -подсвечивает}
                                        {"горячую" клавишу }
        NewSubMenu('~D~ialog', hcNoContext, NewMenu(
            NewItem('~O~pen', 'F3', kbF3, cmDemoDialog, hcNoContext,
            NewLine(
            NewItem('E~x~it', 'Alt-X', kbAltX, cmQuit, hcNoContext,
            nil))),
        nil))
    ));;
end;

{ Статус - строка }
{ В констракторе New(PStatusLine, Init(R,...) программист также
организует вложенность подпунктов NewStatusDef( I,J , ...) с со-
ответствующими NewStatusKey(...) , набор которых будет выпол-
няться лишь в диапазоне номеров контекстов подсказок I,J.
Здесь дана одна статус-строка, реализуемая во всем диапазоне}

procedure TMyApp.InitStatusLine;
var R: TRect;
begin
    GetExtent(R);
    R.A.Y := R.B.Y - 1;
    StatusLine := New(PStatusLine, Init(R,
        NewStatusDef(0, $FFFF,
            NewStatusKey('F10-Menu', kbF10, cmMenu,
            NewStatusKey('~Alt-X~ Exit', kbAltX, cmQuit,
            NewStatusKey('~Alt-F3~ Close', kbAltF3, cmClose,
                nil))),
            nil)
    ));
end;

{ Метод, создающий диалоговое окно }
procedure TMyApp.DemoDialog;
var
    Info: PView;        { Указатель на базовый объект TView    }
    Dialog: PDemoDialog;    { предка TCheckBoxes, }
                            {  TRadioButtons и др.}
    R: TRect;
    C: Word;
begin

{ Последовательность вставки элементов диалогового окна       }
{ Определяет очередность следования полей данных
переменной Data }
{ Невыполнение этого требования вызывает неверное хранение
данных }

{ Data }
R.Assign(20, 6, 60, 22);
Dialog := New(PDemoDialog, Init(R, 'Demo Dialog'));
with Dialog^ do
begin
    {-------------------------------------------}
    R.Assign(3, 3, 20, 6);
    Info := New(PCheckBoxes, Init(R, { Переключатель       }
                                     { независимых кнопок}.
        NewSItem('Monitor ~1~',      { Отмеченные биты     }
                                     { в Data.CheckBoxData}
        NewSItem('Monitor ~2~',      { есть номера         }
                                     { отмеченных кнопок }
        NewSItem('Monitor ~3~',
        nil)))
    ));
    Insert(Info);   { Insert - организует связный список  }
                    { элементов, движение по элементам окна}
                    { поддерживается клавишей Tab}
    R.Assign(2, 2, 18, 3);
    Insert(New(PLabel, Init(R, 'Best magazine', Info))); {PLabel-}
                    {отображает                          }
                    { пояснительный текст, функционально }
                    { связывается с PCheckBoxes          }
                    {-------------------------------------------

R.Assign(22, 3, 38, 6);
Info := New(PRadioButtons, Init(R, { Переключатель зависимых }
                                   { кнопок.                 }
    NewSItem('~W~orld PC', { Дает возможность                }
                           { отметить только                 }
    NewSItem('~D~r. Dobb', { один бит в Data.RadioButtonData. }
    NewSItem('~P~C ',      { Номер этого бита и есть номер}
    nil)))                 { выбранной кнопки }
));
Insert(Info);
R.Assign(21, 2, 33, 3);
Insert(New(PLabel, Init(R, 'Other', Info)));


{-------------------------------------------}
R.Assign(3, 8, 37, 9);
Info := New(PInputLine, Init(R, 34));    { Строка ввода. Введенная }
                                         { информация              }
Insert(Info);                            { хранится в Data.InputLineData }
R.Assign(2, 7, 24, 8);
```

Пример .Продолжение

```
Insert(New(PLabel, Init(R, 'Input name', Info)));
{-------------------------------------------------}
R.Assign(15, 12, 25, 14);
Insert(New(PButton, Init(R, '~O~k', cmOK, bfDefault)));
    { Вставка кнопки "Ok", где: }
    { bfXXX - button flag, равный bfDefault, т.е. нажатие Enter}
    { при любом выбранном элементе меню}
    {зафиксирует изменения в}
    { данных и удалит окно. Кнопка с bfDefault }
    { в диалоговом окне должна }
    { обязательно быть только одна }

 R.Assign(28, 12, 38, 14);
 Insert(New(PButton, Init(R, 'Cancel', cmCancel, bfNormal)));
    { Вставка кнопки "Cancel",}
    { по умолчанию отменяющей все изменения,}
    { произведенные в окне }

        SelectNext(False); { При первоначальном входе }
                    { в диалоговое окно }
                    { делает активным последний }
                    { элемент списка, т.е }
                    { PCheckBoxes , и наоборот True}
                    { активизирует}
 end;               { Cancel }
Dialog^.SetData(Data); { Копирует информацию из Data}
                    { в элементы окна}

 C := DeskTop^.ExecView(Dialog);    { Метод, вызывающий}
                                    { диалоговое окно }
if C<> cmCancel then Dialog^.GetData(Data);
                    { Копирует информацию из окна в Data,}
                    { если не была нажата Cansel, Esc или}
                    { отмечена мышкой закрывающая кнопка }
    Dispose(Dialog, Done); { Вызов дестрактора }
 end;

  var
 MyApp: TMyApp;             { Указатель на объект-программу }

 begin                      { Начало программы          }
    with Data do            { Начальные значения Data   }
    begin
        CheckBoxData := 1;
        RadioButtonData := 0;
        InputLineData:= 'Монитор';
    end;                    { Выполнение }
    MyApp.Init;
    MyApp.Run;
    MyApp.Done;

 end.                                                    ■
```

Как видно из этого примера, процесс "наполнения" объекта TMyApp, потомка Application, достаточно прозрачен. Программу можно дополнить одним из прикладных методов, в котором необходимо проанализировать Data.CheckBoxData, Data.RadioButtonData и Data.InputLineData и выполнить соответствующие их значениям действия. Данный пример иллюстрирует, что трудоемкость по созданию "сервиса" минимальна. Возможно, что после рассмотрения этого примера у читателя останутся "белые пятна" . Это не главное. TV содержит полный набор примеров, позволяющих "с нуля" освоить ООП. Остается решить философский вопрос: сколько времени пришлось бы затратить, чтобы "соорудить" программу, выполняющую то же самое? ■

TEXAS SYSTEMS

НОВАТЕК NOVATECH, Kiev, Ukraine NOVATECH

УВАЖАЕМЫЕ ГОСПОДА!

НОВАТЕК, г.Киев, Украина, совместно с фирмой Universal Research Technologies, Inc. (URT), Хьюстон, Техас, США осуществляет поставку аппаратно-программных комплексов Texas Systems:

- ◇ новейшие модели IBM PC совместимых персональных компьютеров на базе 80386DX/40 МГц, 80486SX/33 МГц, 80486DX/2 50МГц, 80486DX/50МГц;
- ◇ широкий спектр периферийных устройств;
- ◇ последние разработки ведущих производителей программных продуктов США и других стран.

НИЗКИЕ ЦЕНЫ!
Возможна поставка специальных заказных спецификаций.

При поставке продукции проводятся предпродажная подготовка и тестирование каждого устройства по специально разработанной в США системе контроля качества.

Потребители обеспечиваются 12 - месячной гарантией на все типы проданных систем (на периферийные устройства - гарантийные сроки фирм - изготовителей).

Оплата продукции производится в СКВ

Украина, 252030, г. Киев - 30, ул. Ленина, 39 ; 252028, г. Киев - 28, проспект Науки, 7

Телефоны: (044) 224-0471, 225-2282. Телефакс: (044) 267-3975, 224-0022.

Начальная установка обработчика событий в среде Turbo Vision

В. Водолазкий

Статья посвящена обработке событий в Turbo Vision (версия для языка Turbo Pascal 6.0).

Вообще-то говоря, успех шестой версии языка Turbo Pascal в значительной степени объясняется тем, что фирма Borland включила в его состав библиотеку объектно-ориентированных процедур, предназначенную для унификации разработки пользовательского интерфейса. Однако когда начинаешь знакомиться с Turbo Vision поближе, выясняешь, что все это далеко не так просто, как хотелось бы. Конечно тривиальные программы получаются легко и быстро путем переделки поставляемых вместе с библиотекой примеров. Но что делать, если необходимо реализовать функцию, которой в примерах нет, а объектно-ориентированное программирование, если и не тайна за семью печатями, то во всяком случае требует достаточно много времени для его изучения.

В этой статье мы рассмотрим только один маленький аспект программирования в среде Turbo Pascal + Turbo Vision. Автор надеется, что читатель сможет использовать предложенное здесь решение в своей работе.

Речь пойдет о начальной установке обработчика событий. Как вы вероятно знаете, основная концепция, принятая в Turbo Vision, состоит в том, что все изменения в сформированном программой "мире" происходят в результате возникновения тех или иных событий (нажатия на кнопку мыши или клавиатуры, аппаратных прерываний, сигнала от таймера и т.д.). События могут обрабатываться или игнорироваться, одни могут порождать другие, и так до тех пор, пока пользователь не введет команду cmQuit, которая (так же, как и все другие) сформирует соответствующее событие и завершит работу программы.

Но что сделать, чтобы при старте программы автоматически обрабатывалось одно из событий? (Примером подобного события может служить вывод на экран аннотации с названием программы, ее версией, автографом автора и т.д.). Вероятно метод TApplication.Run должен при инициализации иметь уже установленные параметры, сообщающие о наступлении события, которое необходимо обработать.

Ну с обработкой, пожалуй, все понятно. Формируем команду, например cmAbout, и резервируем соответствующую строку в обработчике событий (чтобы не занимать драгоценное место на полосе, автор решил не вводить фрагменты кода в тело программы, а вместо этого привести полный текст (листинг). При этом от вас не требуется вводить формирователь данной команды в меню или статус-строку ("горячие" клавиши), если вам достаточно один раз сформировать и обработать это событие в начале программы.

Теперь от вас требуется только принудительно установить встроенный индикатор событий объекта TApplication (или еще раз напомним его потомка) на обработку необходимого вам события. Для этого необходимо (см. листинг):

- сформировать переменную типа TEvent (событие);
- заполнить ее поля;
- What (вид события) - evCommand (команда);
- Command (код команды) - cmAbout (код необходимой команды);
- InfoPtr (указатель на передаваемые другому событию данные) - nil;
- установить событие в обработчик с помощью метода TApplication.PutEvent(var E : TEvent).

Собственно говоря, метод PutEvent принадлежит родителю TApplication - объекту TProgram, который в свою очередь просто наследует его от своего родителя - объекта TView. Сам по себе PutEvent помещает событие-аргумент в очередь событий, требующих обслуживания, в результате чего это событие будет следующим событием, возвращаемым GetEvent (методом, используемым обработчиком событий HandleEvent).

Однако при работе необходимо помнить следующее:

- приведенный метод установки событий позволяет поставить в очередь только одно (!) событие;
- по умолчанию TView.PutEvent вызывает PutEvent владельца события (то есть для того, чтобы установить обработчик TApplication, вы должны вызывать именно TApplication.PutEvent);
- если же вы перекроете метод PutEvent, вся ответственность за обработку установленного события ложится на вас.

Вот, пожалуй, и все. В заключение хочу лишь добавить, что этот метод вы можете с успехом использовать для порождения цепочки событий. Достаточно в конце обработки "своего" события не вызывать метод ClearEvent(E), а установить новое событие:

```
With E do
    begin
        What    := evCommand; { установка команды }
        Command := cmNext;    { следующая команда }
        InfoPtr := nil;       { поле не используется, }
                              {сбрасываем его}
        end;
PutEvent(E);
```

Пользуйтесь на здоровье! ■

Листинг

```
program AboutDemo;
{$x+}
{
    Программа демонстрации начальной установки событий
    В.В.Водолазкий (C) "Монитор" , 1992, Москва
}

uses objects,drivers,dialogs,views,app;

CONST
    cmAbout = 1000; { код команды вывода идентификатора }

TYPE

    MyApp  = object(TApplication)

        procedure About;
        procedure HandleEvent(var E : TEvent); virtual;

        end;

Procedure  MyApp.About;
    {
    формирование окошка с фирменной меткой
    }
    var     d: PDialog;
            V: PView;
            R: TRect;
    begin
        R.Assign(0,0,40,12) ;{задаем размеры окошка}
        D := New(PDialog, Init(R,'Аннотация'));
        With D^ do
             begin
            Options := Options or OfCentered; {центрируем }
                                        { окошко на экране}
```

Продолжение листинга

```
            R.Grow(-1,-1);              {не трогаем рамку}
            Dec(R.B.Y, 3);
            Insert(New(PStaticText, Init(R,
            #13 +
            ^C'Работаем в Turbo Vision'#13+
            #13 +
            ^C'Версия 1.0'#13+
            #13+
            ^C'В.В.Водолазкий'#13 +
            ^C'Москва (C) 1991')));

    R.Assign(15,9,27,11) ; {окошко для отключения}
    Insert(New(PButton,Init(R,'~П~родолжай',cmOk,bfDefault)))
end;
V := ValidView(D);
  if V <> nil then
    begin
        DeskTop^.ExecView(D);
        Dispose(D,Done);
    end
end;

procedure MyApp.HandleEvent(var E : TEvent);

{
**** собственно процедура обработки прерываний
        от пользователя ***
}

begin
    TApplication.HandleEvent(E); { вызов родительсокого }
                                 { обработчика }
    IF E.What = evCommand then
        begin
            case E.Command of
                cmAbout  : About;
            else
                Exit
            end;
            ClearEvent(E)
    end
end;

VAR
M  :   MyApp;    { экземпляр обьекта приложения }
EE :   TEvent;

BEGIN

    M.Init;

{ — установка стартового состояния Event-ера — }

    With EE do
            begin
                What    := evCommand;
                Command  := cmAbout;
                InfoPtr := nil;
            end;
     M.PutEvent(EE); { устанавливаем начальное событие }

    M.Run;
    M.Done
end.
```

■

Софткит™ выступает «ЗА» Clipper®!

Интервью с участником 3-го Международного Компьютерного Форума, проходившего с 16 по 18 июня в Центре Международной торговли на Красной Пресне в Москве

По мнению редакции, одним из наиболее примечательных событий МКФ стало участие в нем новой отечественной частной фирмы - Корпорации Софткит™ (Softkit™ Corporation).

Корпорация Софткит основана в 1991 году Президентом Корпорации Владимиром Галкиным. Софткит специализируется на разработке программного обеспечения для IBM - совместимых персональных компьютеров.

Основным продуктом Корпорации, который представлялся на форуме, был Инструментальный пакет для системы Clipper - Клипкит™ версии 1.0.

Президент Корпорации Софткит Владимир Галкин любезно согласился дать интервью для нашего журнала.

Монитор. *Какие цели Вы преследовали, принимая участие в МКФ, каковы результаты и Ваши впечатления от его работы?*

В.Галкин. Этот компьютерный форум явился первым официальным выходом Корпорации на рынок в качестве независимого разработчика инструментальных и прикладных программных продуктов преимущественно для системы Clipper. Главная задача, которую мы поставили для себя перед участием в форуме - представить основные направления деятельности Корпорации, состав разрабатываемых программных продуктов и перспективных технологий, стратегию продвижения Корпорации на внутреннем и внешнем рынках - на наш взгляд успешно решена. Хотелось бы отметить исключительно высокий уровень проведения МКФ и одновременно выразить сожаление в связи с отсутствием в составе участников МКФ представителей фирмы Nantucket. Причиной этого, как мне представляется, явилось слияние фирмы Nantucket Corp. с фирмой Computer Associates, Inc., известной как производитель семейства электронных таблиц SuperCalc. Недостаток информации об этом событии вызвал множество самых различных суждений о судьбе и перспективах системы Clipper вообще. Наибольшее удовлетворение доставило мне на МКФ исключительно высокое внимание посетителей, которого удостоился наш стенд, и то, что учитывая отсутствие представителей фирмы Nantucket на МКФ, мы оказались чуть ли не единственными представителями интересов направления системы Clipper на форуме.

М. *Осуществляются ли Ваши замыслы?*

В.Г. Можно сказать с уверенностью "Да". Организационные и финансовые проблемы, которые стоят на пути продвижения фирм, подобных нашей Корпорации, в специфических условиях Российской действительности, пока удается успешно решать вопреки всему. В настоящее время происходит интенсивный рост Корпорации и объединение усилий различных фирм-разработчиков программного обеспечения вокруг проектов и работ, проводимых совместно с Корпорацией.

М. *Что вы можете сказать о ситуации, складывающейся на рынке программных продуктов для IBM - совместимых персональных компьютеров у нас в стране?*

В.Г. Происшедшая за последнее время перегруппировка сил среди ведущих производителей программных продуктов обещает в ближайшем будущем значительное усиление конкурентной борьбы. Складываются три основные конкурентные линии, которые будут определять состояние рынка на ближайшую перспективу: FoxPro фирмы Fox Software Inc. в союзе с фирмой Microsoft, dBASE и Paradox в составе фирмы Borland и Clipper фирмы Nantucket в объединении с фирмой Computer Associates. Происходящие за рубежом процессы оказывают влияние и на состояние рынка программных продуктов внутри страны, и это влияние необходимо учитывать. Корпорация Софткит выступает за практическую поддержку и дальнейшее развитие технологий на базе системы Clipper.

М. *В связи с упомянутыми Вами процессами перегруппировки сил, как Вы оцениваете перспективы распространения системы Clipper в России?*

В.Г. Несмотря на ряд имеющихся проблем, я считаю, что перспективы у системы Clipper в стране очень хорошие. Конечно многое зависит от того, как будет действовать фирма Nantucket в своем новом состоянии, каким изменениям подвергнется стратегия поддержки системы Clipper, будут ли учтены недостатки в проведении своей линии представителем фирмы Nantucket в лице СП "Магнит" и от ряда других моментов. Относительно СП "Магнит" можно было бы высказать пожелание более активно поддерживать независимых разработчиков инструментальных программ для системы Clipper и освободиться от некоторого налета "академичности" в работе с пользователями. Главная идея работы с пользователями должна быть понятна и доступна - система Clipper это продукт для всех, а не для "касты избранных", и мы с помощью наших инструментальных средств постараемся доказать это.

М. *Как Вы оцениваете ситуацию, складывающуюся сейчас в нашей стране для разработчиков и пользователей системы Clipper?*

В.Г. Clipper очень распространен у нас в стране и потенциал разработчиков прикладных программ обеспечивает прочные позиции этой технологии. Несмотря на наличие мощных средств разработки прикладных систем, имеющихся в составе базового комплекта Clipper, разработчики сталкиваются с проблемой эффективного использования всех потенциальных возможностей этих средств.

Представляется не естественным, что сегодня программисты вынуждены либо разрабатывать для себя инструментальные средства самостоятельно, при условии наличия времени и средств, либо отказываться от части имеющихся у Clipper возможностей. В этом мы видим одно из слабых мест системы Clipper по сравнению с конкурирующими продуктами, где инструментальное окружение представлено особенно сильно. Создание мощного конкурентноспособного инструментального окружения для системы Clipper на основе современных технологий и тенденций программирования является основным направлением деятельности нашей Корпорации на ближайшую перспективу. Наш инструментальный пакет Клипкит версии 1.0 - первый из продуктов этого направления, который выводится на рынок. Мы уверены, что сочетание системы Clipper с хорошими инструментальными средствами резко повысит эффективность и качество труда разработчиков прикладных систем.

М. *Расскажите подробнее о Вашем инструментальном пакете для Clipper - Клипкит версии 1.0.*

В.Г. Инструментальный пакет Клипкит версии 1.0 представляет собой расширение базового комплекта Clipper и Clipper Tools One (Summer 87) набором из более чем 500 готовых для использования функций, которые позволяют решать большинство проблем организации эффективного интерфейса пользователя в прикладных программных продуктах. Содержание пакета во многом соответствует инструментальным пакетам других фирм, однако именно отсутствие этих средств ставит разработчиков программ на базе системы Clipper в неравные условия по отношению к конкурентам. Первая версия пакета реализует традиционную технологию программирования. Следующие версии строятся полностью на технологии объектно-ориентированного программирования (ООП). Они реализуются в виде готовых для применения конструкций и механизмов, позволяющих прикладным программистам легко и надежно создавать эффективные и конкурентноспособные программные продукты любой сложности с организацией пользовательского интерфейса на уровне стандартов лучших образцов программных продуктов (Norton Commander, Microsoft Windows и др.) для псевдографической среды программирования системы Clipper. Нами разработана собственная технология применения методологии ООП и для Clipper версии Summer 87, и для Clipper версии 5.0, которая реализуется в дополнение и независимо от аналогичной линии самой фирмы Nantucket. Реализация этого подхода предполагает создание библиотек встроенных КЛАССОВ для обеих версий. На стенде демонстрировалась реализация технологии ООП на примере одного из встроенных КЛАССОВ - КЛАССА DIR. Этот КЛАСС позволяет создавать и манипулировать объектами в виде окна, воспроизводящего интерфейс директорий Norton Commander 3.0. Применение ООП технологии приводит к существенному улучшению характеристик прикладных программ, а также ускоряет сам процесс разработки.

М. *Каковы на Ваш взгляд основные проблемы, стоящие перед системой Clipper, которые могут отразиться на его борьбе с конкурентами?*

В.Г. Обоснованное беспокойство вызывает ситуация, связанная с некоторыми нестыковками последних версий Microsoft C 6.00 и 7.00 с Clipper 5.0. Предложение, высказанное представителями фирмы Nantucket, об использовании в разработках Microsoft C 5.1 вполне понятно, но не вселяет оптимизма, так как это означает отставание на уровне базовых средств и технологий по сравнению с ведущими конкурентами. Логичным решением этой проблемы, как мне представляется, было бы объединение усилий фирмы Nantucket по разработке Clipper 6.0 со средствами и технологиями другой конкурирующей в этом классе программных средств фирмы - Jensen and Partners UK, что позволило бы Clipper 6.0 успешно конкурировать на равных с его ближайшим соперником - системой FoxPro в союзе с фирмой Microsoft.

М. *Каковы ближайшие планы деятельности Корпорации?*

В.Г. В конце сентября 1992 года инструментальный пакет Клипкит версии 1.0 для Clipper и Clipper Tools One (Summer 87) поступит в распространение первоначально по предварительным заказам, а затем в свободную продажу. В конце года будет подготовлена для реализации версия 2.0 для Clipper и Clipper Tools One версии 5.0. Корпорация планирует на ближайшее будущее поддержку и развитие этих двух версий. Мы собираемся принять участие в предстоящей выставке Softool-92 в октябре 1992 года, где будут демонстрироваться сам пакет в полной комплектации, перспективные технологии, используемые при разработке следующих версий пакета, а также ряд наших прикладных программных продуктов.

М. *Какие перспективные технологии получат наибольшее развитие в Ваших продуктах?*

В.Г. ООП методология, реализованная нами для обеих версий системы Clipper, позволяет использовать наряду с многооконными интерфейсами технологию виртуальных объектов на базе специально разработанного нами механизма вытеснения объектов из оперативной памяти на диск и обратно - Virtual Object Manager (VOM). Скрытое для программиста использование этого механизма при работе с классами дает возможность создания большого количества объектов на ограниченном пространстве динамической памяти и таким образом открывает широкие перспективы применения ООП технологии для системы Clipper.

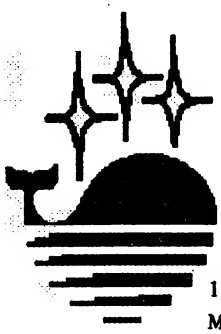
М. *Какие инструментальные средства Вы предполагаете распространять в составе пакета Клипкит?*

В.Г. В пакет будут включены полезные при разработке прикладных систем утилиты, среди которых система защиты от несанкционированного копирования исполняемых файлов, программы-упаковщики, средства восстановления поврежденных файлов баз данных, программы-конверторы баз данных и другие утилиты.

М. *Какие прикладные программные продукты планируются для разработки и распространения?*

В.Г. В рамках развития линии системы Clipper предполагается на базе предлагаемых инструментальных средств реализовать электронную таблицу с открытым для программистов внутренним интерфейсом (табличные инструментальные функции), графическими возможностями, аппаратом математической статистики и т.д. Наряду с этим большим проектом идет работа по созданию нескольких прикладных систем, среди которых Инструментальная среда разработчика, Экспертная система для анализа биржевой информации , Экспертная система обучения английскому языку по стандарту "TOEFL". Новые биржевые технологии и другие инструментальные средства и технологии.

Интервью брал Дмитрий Бройтман.



К о р п о р а ц и я

С о ф т к и т

S o f t k i t

C o r p o r a t i o n

117607, Москва, тел.(095)932-69-13
Мичуринский пр-т, 31/1 (095)931-79-33

Предприятиям, Организациям и Фирмам

Пользователям системы КЛИППЕР

Корпорация С о ф т к и т

представляет

ТУЛКИТ для КЛИППЕРа

Инструментальный пакет программ для профессиональных пользователей системы КЛИППЕР

К Л И П К И Т

Корпорация Софткит, Версия 1.0, 1992 г., Москва

Инструментальный пакет **"К л и п к и т"** Версия 1.0

Представляет собой мощное инструментальное средство для профессиональных программистов - разработчиков прикладных систем и АРМов на базе КЛИППЕР.

Пакет включает библиотеку программ с более чем 500 пользовательскими функциями и рядом сервисных утилит, которые обеспечивают функционально полный набор интерфейсных средств для КЛИППЕРа, существенно расширяющих и упрощающих использование его базовых возможностей, повышающих качество разрабатываемых прикладных систем.

Пакет включает следующие основные направления средств:

- ОКНА
- МЕНЮ
- СООБЩЕНИЯ
- УСТРОЙСТВА
- МЫШЬ, КУРСОР
- УПРАВЛЕНИЕ ПАМЯТЬЮ
- КЛАССЫ ОБЪЕКТОВ
- ГРАФИКУ
- КОММУНИКАЦИИ
- СЕРВИС

"В ХОРОШИХ ИНСТРУМЕНТАХ НУЖДАЮТСЯ ВСЕ, НО В ПЕРВУЮ ОЧЕРЕДЬ - ВЕЛИКИЕ МУЗЫКАНТЫ"
Страдивари

Инструментальный пакет "К л и п к и т" Версия 1.0

Условия поставки

Первая партия пакета в полной комплектации поступит в распространение в конце третьего квартала 1992 г.

Цена пакета - 9000 руб. + 28% НДС.

Распространение пакета осуществляется первоначально по предварительным заказам, а затем в свободной продаже. Оформление предварительного заказа дает право на 20% скидку к цене пакета независимо от количества комплектов в заказе (минимальный объем заказа для организаций - 2 комплекта).

Заказ на приобретение пакета партиями (10 и более штук) дополнительно дает право на следующие скидки к цене пакета:
от 10 до 50 - 10%;
от 50 до 100 - 20%;
от 100 и более - 30%.

По желанию заказчика в комплект поставки может быть включена в соответствии с диллерским договором система Клиппер (Clipper Summer 87 & Clipper Tools One) за отдельную плату в размере 10 000 руб + 28% НДС. При этом указанные выше правила скидок на поставку системы Клиппер не распространяются!

Для оформления предварительного заказа необходимо направить по адресу Корпорации бланк заказа (письмо от организации) с указанием комплектации и количества комплектов, обратного почтового адреса и копии документа о перечислении предоплаты в соответствии с отмеченными выше условиями на первую партию пакета в течение июня-июля 1992 г.

Приобретение партий пакета для последующего распространения осуществляется только на основе заключенного диллерского договора.

С момента поступления предоплаты на счет заказ считается оформленным. Выполнение заказа может быть осуществлено как на территории Корпорации по предварительным приглашениям, высылаемым по адресу заказчика, так и пересылкой пакета почтой.

Для формирования предварительных заказов и более полного информирования потенциальных пользователей о возможностях пакета и технологии разработки программ с его применением в настоящее время подготовлены и распространяются демонстрационный (DEMO) и учебный (TUTOR) комплекты пакета, являющиеся дополнительными составляющими полной комплектации пакета, распространяемыми независимо.

Приобретение этих комплектов позволяет сразу приступить к практическому освоению технологий использования пакета в прикладных разработках и принять обоснованное решение о целесообразности оформления предварительного заказа на полную версию пакета.

Цена демонстрационного комплекта (DEMO) - 300 руб. + 28% НДС.

Цена учебного комплекта (TUTOR) - 600 руб. + 28% НДС.

Приобретение этих комплектов дает право на следующие скидки к цене пакета: демонстрационный комплект(DEMO) - 10%; учебный комплект(TUTOR) - 20%.

Зарегистрированные пользователи получают право на приобретение всех последующих версий пакета с 25% скидкой к цене пакета, пользоваться консультациями и методической поддержкой специалистов Корпорации.

В состав полной комплектации пакета входят:

1. Документация(250 стр.), содержащая:
- руководство пользователя;
- справочное руководство;
- учебное пособие по приложениям.

2. Лицензионное соглашение.

3. Регистрационная карточка.

4. Четыре дистрибутивные дискеты (типа DS/DD,360K) с архивированной информацией(.LZH), включающие:
Диск#1 - инстал/Утилиты;
Диск#2 - библиотеки;
Диск#3 - справочник(NG);
Диск#4 - приложения.

В состав демонстрационного комплекта (Диск DEMO) входят:

1. Документация, которая содержит рекламные проспекты, инструкции, описания примеров, бланки заказа на полный комплект пакета.

2. Одна дистрибутивная дискета (типа DS/DD, 360K) с информацией в архивированном виде (.LZH), содержащей инструкции и программу инсталяции, исполняемые демонстрационные программы (.EXE), исходные тексты программ (.PRG) с комментариями, файлы демонстрационных баз данных и примеров.

В состав учебного комплекта (Диск TUTOR) входят:

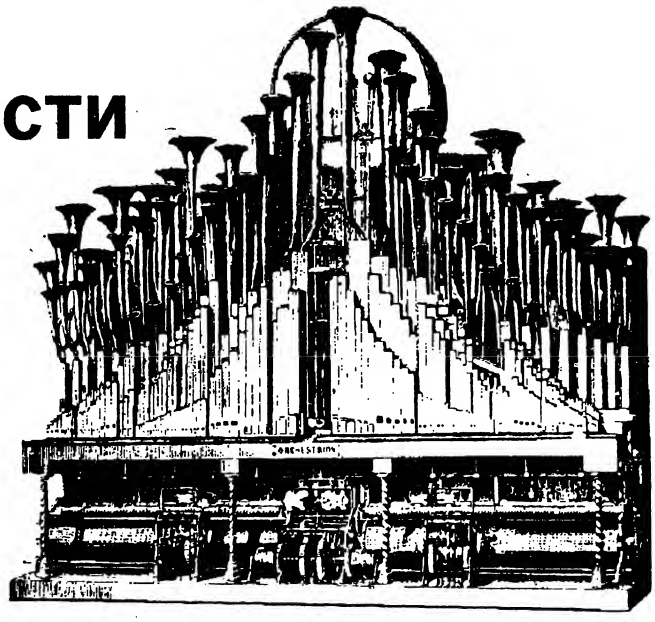
1. Документация, которая содержит рекламные проспекты, инструкции, описания примеров, лицензионное соглашение на использование библиотек учебного пакета, бланки заказа на полный комплект пакета.

2. Одна дистрибутивная дискета(типа DS/DD, 360K) с информацией в архивированном виде(.LZH), содержащей инструкции и программу инсталяции, учебную библиотеку пакета с более чем 100 пользовательскими функциями (.LIB), утилиты, исходные тексты демонстрационных программ (.PRG)., файлы примеров.

ЗАКАЗЫ НАПРАВЛЯТЬ ПО АДРЕСУ: 117607, Москва, Мичуринский пр-т, 31/1, ТМО
БАНКОВСКИЕ РЕКВИЗИТЫ: р/с 006468455 в МАБ "Темпбанк" корр./сч.161104 в РКЦ ГУ ЦБ РФ в г. Москве, МФО 201791

Clipper 5.0 – новые возможности

А. Чернышков



Часть 4

Программные конструкции

Статью с таким заголовком я вначале представлял себе как набор выдержек из разных программ , раскрывающих приемы программирования на Clipper 5.0 . Однако при поиске примеров в своей библиотеке я натолкнулся на несколько связанных одной целью коротких программ , которые могут проиллюстрировать (как мне кажется) наиболее яркие возможности языка, а именно :

- использование STATIC и LOCAL переменных ;
- определение и использование объектов сложной структуры ;
- использование блока кода , передаваемого в функцию в качестве параметра ;
- обеспечение возможности рекурсивного вызова функций ;
- переопределение стандартных команд.

Представленный комплекс функций , находящихся в файле Boxes.prg (листинг 1), я назвал подсистемой работы с боксами (псевдоокнами) , которая в некоторой степени имитирует оконный интерфейс с пользователем.

Функция BoxNew() создает массив со структурой , описанной в начале файла Boxes.prg. Такой массив можно представить как объект сложной структуры . Каждый элемент массива хранит одну из характеристик этого объекта . Функция BoxNew() возвращает в вызывающую программу адрес объекта , который можно присвоить какой-либо переменной . В нашем случае эту переменную можно назвать объектом типа "бокс" . Функции BoxOpen() и BoxClose() определяют два действия , которые можно производить с обьектом типа "бокс" . Как видно из названий функций , одна из них "открывает" бокс (выводит его на экран) , а другая - закрывает (убирает бокс с экрана) .

Четыре STATIC переменных , объявленных в начале файла Boxes.prg , будут определять координаты текущего бокса , выведенного на экран функцией BoxOpen() . Доступ к этим переменным из внешних программ реализуется с помощью функций RowTop() , ColLeft() , RowBottom() , ColRight() и SetBoxScreen() . Функция BoxOpen() кроме адреса бокса может принимать еще два параметра . С помощью параметра cColor можно переопределить цвет , установленный ранее при создании бокса . Параметр bBlock позволяет реализовать конструкцию типа "выполнить такой-то процесс в таком-то боксе". В этом случае после выполнения процесса бокс автоматически закрывается . Возможность рекурсивного вызова функции BoxOpen() обеспечивается тем , что все характеристики бокса, а также координаты предыдущего открытого бокса, хранятся в массиве , описывающем бокс .

Для вывода в бокс информации с привязкой к его координатам служит файл Boxin.ch (листинг 2) , который реализует переопределение команд SAY и GET таким образом , что за строку и колонку с номером 0 будут приниматься строка и колонка не экрана, а бокса . Отмена переопределения производится включением в программу файла Boxout.ch (листинг 3).

Приведем небольшой пример использования подсистемы работы с боксами. Предположим, что нам необходимо из файла наших клиентов "откачать" во временный файл записи для какого-либо одного региона. Программа MOVE_CLI.PRG (листинг 4) реализует это для исходного файла с именем CLIENT.DBF , в котором имеется поле REGION для указания города местонахождения клиента , и поле NAMECLIENT - наименование клиента. Файл CLIENT.DBF для тестирования программы MOVE_CLI.PRG можно создать с помощью программы CRCLIENT.PRG (листинг 5). ■

Листинг 1

```
/***
Программа : Boxes.prg
Версия : Clipper 5.01
Автор: Чернышков А.С.
Подсистема работы с боксами
Компиляция:clipper boxes /m/n/b
*/

#define SEP_LEFT''              && Разделитель заголовка левый
#define SEP_RIHT''              && Разделитель заголовка правый

// Структура объекта "бокс"

#define BX_LEN          12      && Длина массива структуры бокса
#define BX_ROWH         1       && Строка верхняя
#define BX_COLL         2       && Колонка левая
#define BX_ROWB         3       && Строка нижняя
#define BX_COLR         4       && Колонка правая
#define BX_LSAVE        5       && .T. - сохранять экран
                                && при вызове бокса
#define BX_LASTCOLOR    6       && Старый цвет
#define BX_BOX          7       && Символы рамки
#define BX_SHADOW       8       && .T. - с тенями
#define BX_COLOR        9       && Цвет бокса
#define BX_AREASCR      10      && Область сохранения экрана
#define BX_TYTLE        11      && Заголовок
#define BX_LASTBOX      12      && Массив координат
                                && предыдущего бокса

STATIC RowTop := 0
STATIC ColLeft := 0
STATIC RowBottom := 24
STATIC ColRight := 79

/*
    BoxNew(nRowH,nColL,nRowB,nColR, cBox,cColor,
                lSaveScreen,lShadow,cTytle) --> aBox
Создает новый бокс
*/

FUNCTION BoxNew(nRowH,nColL,nRowB,nColR,Box,cColor,
                                lSaveScreen,lShadow,cTytle)

 LOCAL aBox[BX_LEN]
 aBox[BX_ROWH] := iif(nRowH = = NIL, 0,nRowH)
 aBox[BX_COLL] := iif(nColL = = NIL, 0,nColL)
 aBox[BX_ROWB] := iif(nRowB = = NIL,MaxRow(),nRowB)
 aBox[BX_COLR] := iif(nColR = = NIL,MaxCol(),nColR)
 aBox[BX_LSAVE] := iif(lSaveScreen = = NIL, F., lSaveScreen)
 aBox[BX_LASTCOLOR] := ""
aBox[BX_COLOR] :=  IIF( cColor = = NIL, setcolor(),  cColor )
 aBox[BX_BOX] := iif(cBox = = NIL,'',cBox)
 aBox[BX_SHADOW] := iif(lShadow = = NIL,.F.,lShadow)
 aBox[BX_AREASCR] := ""
 aBox[BX_TYTLE] := cTytle

 RETURN aBox

 /*
 BoxOpen( aBox [,cColor] [,bBlock] ) --> aBox
 Выводит бокс на экран
 */

 FUNCTION BoxOpen(aBox,cColor,bBlock)
 LOCAL ColTytle
  aBox[BX_LASTBOX] := SetBoxScreen()
  if aBox[BX_LSAVE]
      aBox[BX_AREASCR] := ;
          savescreen(aBox[BX_ROWH],;
                     aBox[BX_COLL],;
                     aBox[BX_ROWB]+1,;
                     aBox[BX_COLR]+1)
 endif

 aBox[BX_LASTCOLOR] := if(ValType(cColor) = = 'C',;
                                 setcolor(cColor),;
                                 setcolor(aBox[BX_COLOR]))
 @ aBox[BX_ROWH],;
    aBox[BX_COLL],;
    aBox[BX_ROWB],;
    aBox[BX_COLR] ;
    box aBox[BX_BOX]

 if aBox[BX_SHADOW]
          BoxShadow(aBox[BX_ROWH],;
          aBox[BX_COLL],aBox[BX_ROWB],aBox[BX_COLR])
 endif
 if aBox[BX_TYTLE] ! = NIL
          ColTytle := ((aBox[BX_COLR] - aBox[BX_COLL]) - ;
          len(aBox[BX_TYTLE]))/2 + aBox[BX_COLL]
          @ aBox[BX_ROWH],ColTytle say ;
          SEP_LEFT+aBox[BX_TYTLE]+SEP_RIHT
 endif
 SetBoxScreen(aBox[BX_ROWH]+1,
 aBox[BX_COLL]+1,;
 aBox[BX_ROWB]-1,aBox[BX_COLR]-1)
 SetPos(aBox[BX_ROWH]+1,aBox[BX_COLL]+1)
 f ValType(bBlock) = = 'B'
      eval(bBlock)
      BoxClose(aBox)
 endif
 RETURN aBox

 /*
      BoxClose(aBox) --> aBox
      Деактивирует бокс (и убирает бокс с экрана,если при создании
      был указан параметр lSaveScreen)
 */

 FUNCTION BoxClose(aBox)
       if .not. empty(aBox)
            if aBox[BX_LSAVE]
                 restscreen(aBox[BX_ROWH], aBox[BX_COLL],;
                            aBox[BX_ROWB]+1, aBox[BX_COLR]+1,;
                            aBox[BX_AREASCR])
      aBox[BX_AREASCR] := ""
 endif
 setcolor(aBox[BX_LASTCOLOR])
 SetBoxScreen(aBox[BX_LASTBOX])
 endif
 RETURN aBox

 /*
 SetBoxScreen(RowTop,ColLeft,;
              RowBottom,ColRight) --> aLastScreen
 Устанавливает RowTop,ColLeft,RowBottom,ColRight
 в соответствии с координатами бокса
 Возвращает массив со старыми значениями:
 верхняя строка ,левая колонка,нижняя строка, правая колонка
 */

 FUNCTION SetBoxScreen(RowTop,ColLeft,;
                       RowBottom,ColRight)
 LOCAL aLastScreen[4]
 if ValType(RowTop) = = 'A'
   do while len(RowTop) < 4
       aadd(RowTop,NIL)
   enddo
     aLastScreen[1] := if(RowTop[1] = = NIL, RowTop(), RowTop(RowTop[1]))
     aLastScreen[2] := if(RowTop[2] = = NIL,ColLeft(), ColLeft(RowTop[2]))
     aLastScreen[3] := if(RowTop[3] = = NIL,RowBottom(),;
                          RowBottom(RowTop[3]))
    aLastScreen[4] := if(RowTop[4] = = NIL, ColRight(), ColRight(RowTop[4]))
 else
     aLastScreen[1] := if(RowTop   = = NIL, RowTop(), RowTop(RowTop))
     aLastScreen[2] := if(ColLeft  = = NIL, ColLeft(), ColLeft(ColLeft))
     aLastScreen[3] := if(RowBottom = = NIL, RowBottom(),;
                          RowBottom(RowBottom))
     aLastScreen[4] := if(ColRight  = = NIL, ColRight(), ColRight(ColRight))
   endif
 RETURN aLastScreen



 FUNCTION RowTop(nRow)
    LOCAL n := RowTop
   if nRow < > NIL
```

```
      RowTop := nRow
   endif
RETURN n

FUNCTION ColLeft(nCol)
   LOCAL n := ColLeft
   if nCol <> NIL
      ColLeft := nCol
   endif
RETURN n

FUNCTION RowBottom(nRow)
   LOCAL n := RowBottom
   if nRow <> NIL
      RowBottom := nRow
   endif
RETURN n

FUNCTION ColRight(nCol)
   LOCAL n := ColRight
   if nCol <> NIL
      ColRight := nCol
   endif
RETURN n

/*
 BoxShadow( <nTop>, <nLeft>, <nBottom>, <nRight> ) --> NIL
 Нарисовать тени
*/
FUNCTION BoxShadow( nTop, nLeft, nBottom, nRight )

LOCAL nShadTop, nShadLeft, nShadBottom, nShadRight
nShadTop := nShadBottom := MIN(nBottom + 1, MAXROW())
nShadLeft := nLeft + 1
nShadRight := MIN(nRight + 1, MAXCOL())
RESTSCREEN( nShadTop, nShadLeft,;
            nShadBottom, nShadRight,;
     TRANSFORM( SAVESCREEN(nShadTop, nShadLeft,;
                           nShadBottom, nShadRight),;
     REPLICATE("X", nShadRight - nShadLeft + 1)))
nShadTop := nTop + 1
nShadLeft := nShadRight := MIN(nRight + 1, MAXCOL())
nShadBottom := nBottom
RESTSCREEN( nShadTop, nShadLeft, nShadBottom, nShadRight,;
     TRANSFORM( SAVESCREEN(nShadTop, nShadLeft ,;
                           nShadBottom, nShadRight),;
          REPLICATE("X", nShadBottom - nShadTop + 1) ) )

RETURN NIL

* End Boxes.prg
```

Листинг 2

Файл Boxin.ch

```
// Подключение этого файла позволяет
// производить вывод информации по
// командам SAY и GET с ориентировкой
// на координаты верхней левой
// точки открытого бокса, сделанного
// функцией BoxNew()

#command @ <row>, <col> SAY <xpr>      ;
                        PICTURE <pic>  ;
         => ;
         DevPos(RowTop() + <row>,      ;
                ColLeft()+ <col> );    ;
         DevOutPict( <xpr>, <pic> )

#command @ <row>, <col> SAY <xpr>      ;
         => ;
         DevPos(RowTop() + <row>,      ;
                ColLeft()+ <col> );    ;
         DevOut( <xpr> )

#command @ <row>, <col> GET <var>      ;
                        [PICTURE <pic>] ;
                        [RANGE <lo>, <hi>] ;
         => ;
         DevPos(RowTop() + <row>,      ;
                ColLeft()+ <col> );    ;
         AAdd( GetList,                ;
               _GET_(<var>,<(var)>,<pic>, ;
               {|_1,_2| RangeCheck(_1,_2, ;
                        <lo>,<hi>)}))  )

#command @ <row>, <col> GET <var>      ;
                        [PICTURE <pic>] ;
                        [VALID <valid>] ;
                        [WHEN <when>]  ;
         => ;
         DevPos(RowTop() + <row>,      ;
                ColLeft()+ <col> );    ;
         AAdd( GetList,                ;
               _GET_(<var>,<(var)>,<pic>, ;
               <{valid}>, <{when}>))

#command @ <row>, <col> SAY <sayxpr>   ;
                        [<sayClauses,...>] ;
                        GET <var>      ;
                        [<getClauses,...>] ;
         => ;
         @ <row>, <col> SAY <sayxpr>   ;
                        [<sayClauses>] ;
         ; @ Row()-RowTop(),           ;
           Col()+1-ColLeft() GET <var> ;
                        [<getClauses>]

#command @ <row>, <col> PROMPT <prompt> ;
                        [MESSAGE <msg>] ;
         => __AtPrompt(RowTop() + <row>, ;
                       ColLeft()+ <col>, ;
                       <prompt>, <msg> )
```

Листинг 3

Файл Boxout.ch

```
// Подключение этого файла позволяет
// отменить установки "boxin.ch"
// и производить вывод информации
// по командам SAY и GET
// с ориентировкой на координаты
// верхней левой точки экрана

#command @ <row>, <col> SAY <xpr>      ;
                        PICTURE <pic>  ;
         => ;
         DevPos(<row>, <col> );        ;
         DevOutPict( <xpr>, <pic> )

#command @ <row>, <col> SAY <xpr>      ;
         => ;
         DevPos(<row>,<col> );         ;
         DevOut( <xpr> )

#command @ <row>, <col> GET <var>      ;
                        [PICTURE <pic>] ;
                        [RANGE <lo>,<hi>] ;
         => ;
         SetPos(<row>, <col>);         ;
         AAdd(GetList,                 ;
              _GET_(<var>, <(var)>, <pic>, ;
              {|_1, _2| RangeCheck(_1, _2, ;
                       <lo>, <hi>)}))

#command @ <row>, <col> GET <var>      ;
                        [PICTURE <pic>] ;
                        [VALID <valid>] ;
                        [WHEN <when>]  ;
         => ;
         SetPos(<row>, <col>);         ;
         AAdd( GetList,                ;
              _GET_(<var>, <(var)>, <pic>, ;
                    <{valid}>, <{when}>))
```

```
#command @ <row>, <col> SAY <sayxpr>     ;
                              [<sayClauses,...>] ;
                              GET <var>          ;
                              [<getClauses,...>] ;
          = >                                    ;
          @ <row>, <col> SAY <sayxpr>            ;
                              [<sayClauses>]
          ; @ Row(), Col()+1 GET <var>           ;
                              [<getClauses>]


#command @ <row>, <col> PROMPT <prompt>;
                              [MESSAGE <msg>] s;
          = > __AtPrompt( <row>, <col>,      ;
                              <prompt> , <msg> )
```

Листинг 4

```
/*
Программа    : MOVE_CLI.PRG
Версия       : Clipper 5.01

 Создание файлов клиентов по регионам из исходного файла CLIENT.DBF.
Программа иллюстрирует использование подсистемы работы с боксами.
 Компиляция и компоновка :

     clipper move_cli /m/n/b
     rtlink fi move_cli,boxes /pll:base50

 ( Файл boxes.prg должен быть
предварительно откомпилирован )
*/

// Рамка из одинарной линии
#define B_SINGLE  ( CHR(218) + CHR(196) + ;
                    CHR(191) + CHR(179) + ;
                    CHR(217) + CHR(196) + ;
                    CHR(192) + CHR(179) )

// Рамка из двойной линии
#define B_DOUBLE  ( CHR(201) + CHR(205) + ;
                    CHR(187) + CHR(186) + ;
                    CHR(188) + CHR(205) + ;
                    CHR(200) + CHR(186) )

PROCEDURE MoveClient
     LOCAL mRegion, NameFile
     // создать бокс
     LOCAL aBox := BoxNew(1,20,4,60,;
           B_SINGLE+' ',"N/G,N/W",.F.,.T.)
     set color to W/B
     clear
     use CLIENT new  && открыть файл клиентов
     BoxOpen(aBox)   && открыть бокс
     do while .T.
          mRegion := space(len(CLIENT->REGION))
          NameFile := space(8)
          // ввести наименование региона и файла
          #include "boxin.ch"
          @ 0,1 say "Регион"  get mRegion
          @ 1,1 say "Переписать в файл " get NameFile
          #include "boxout.ch"
          read
          //
          if (lastkey() = = 27 .or. ;
               empty(mRegion)  .or. ;
               empty(NameFile))
               exit
          else
               // в боксе выполнить перепись
               // записей по региону
               BoxOpen(BoxNew(10,5,13,75,;
                    B_DOUBLE+' ',"N/BG",;
                    .T.,.T.,;      "Перепись в файл "+NameFile),,;
                    {|| CLIENT->;
                              (MoveRegion(alltrim(mRegion),NameFile))})
          endif
     enddo
          BoxClose(aBox)      && закрыть бокс
          CLOSE DATA
RETURN

/*
     Создание файла и перемещение
     в него записей по региону
*/

STATIC FUNCTION MoveRegion(mRegion,NameFile)
     dbCreate(NameFile,dbStruct())
     use (NameFile) new
     CLIENT->(dbEval({|| MoveRec(NameFile)},;
          {|| upper(mRegion) $ ;
               upper(CLIENT->REGION)}))
     use
RETURN NIL

/*
     Перемещение одной записи
*/

STATIC FUNCTION MoveRec(NameFile)
     LOCAL i
     // вывести наименование клиента в бокс
     #include "boxin.ch"
     @ 0,1 say CLIENT->NameClient
     #include "boxout.ch"
     // добавить запись
     (NameFile)->(dbAppend())
/    // переписать содержимое всех полей
     for i := 1 to fcount()
          (NameFile)->(fieldput(i,CLIENT->(fieldget(i))))
     next
RETURN NIL

* End MOVE_CLI.PRG
```

Листинг 5

```
/*
Программа    : CRCLIENT.PRG
Версия       : Clipper 5.01

 Создание исходного файла клиентов  CLIENT.DBF для тестирования
программы MOVE_CLI.PRG .

 Компиляция и компоновка :

     clipper crclient /m/n/b
     rtlink fi crclient /pll:base50

*/

PROCEDURE CrClient LOCAL i
     // создать файл
     dbCreate("CLIENT",;
          {{"NAMECLIENT","C",60,0},{"REGION",  "C",20,0}})
     use CLIENT new

     // заполнить файл отладочной информацией
     for i := 1 to 60
     dbAppend()
     FIELD->NAMECLIENT := str(i,2)+"-й клиент"
     do case
          case i < 21
               FIELD->REGION := "Москва"
          case i < 41
               FIELD->REGION := "Киев"
          case i < 61
               FIELD->REGION := "Минск"
     endcase
  next
     use
RETURN
* End CRCLIENT.PRG
```

Clipper или Paradox : попытка аргументированного выбора

С. Крапивенко

Часть 1

Ниже под Paradox подразумевается СУБД Paradox 3.5, а под Clipper - Clipper Summer'87 (Clipper 4.0), но в отдельных случаях будут даны пояснения применительно к версии Clipper 5.0

1. СРАВНЕНИЕ В ЦЕЛОМ

Из сравнения основных характеристик систем (табл. 1) видно, что функционально Paradox значительно мощнее, чем Clipper. Как будет показано ниже, Paradox удобен и пользователям, и разработчикам, а благодаря большому числу удачных системных решений Paradox уже имеет богатую аудиторию пользователей и продолжает ее расширять.

Clipper обычно используется разработчиками высокой квалификации, которых интересует прежде всего мощность и развиваемость языка программирования, открытость структур данных и архитектуры системы, возможность подключения разноязыких модулей.

По ценам (табл. 2 и 3, без учета НДС, по состоянию на май 1992 г.) пакеты сопоставимы, хотя суммарная стоимость может сильно отличаться.

Например, минимальный комплект для разработчика - пакет для разработки Paradox, среда выполнения RunTime (передается пользователям вместе с приложениями), сетевая поддержка одного пользователя LanPack: 19 900 + 2 600 + 7 200 = 29 700 (руб.). Для пяти одновременно работающих пользователей цена будет уже 58 500 руб. Существенно можно сэкономить: купив библиотеку Paradox Engine 2.0 за 7 200 руб., вы получаете право на разработку и неограниченное распространение приложений, программируя на языке Borland C или Microsoft C и Turbo Pascal (без учета их стоимости).

Разработчикам приложений на Clipper за версию 4.0 и библиотеку CT1 потребуется отдать 10 000 руб., а за тот же набор для версии 5.0 - 39 000 руб.

Если у тебя спрошено будет: что полезнее, солнце или месяц? - ответствуй: месяц. Ибо солнце светит днем, когда и без того светло; а месяц - ночью.

Козьма Прутков. "Мысли и афоризмы", 51.

1.1. PARADOX

Paradox - это мощная, высокопроизводительная система [Кэлман 91], [Чепел 89], удобная и эффективная как в интерактивной работе, так и в разработке приложений [Ганелин 91], [Хан-Магометов 91], [Браун 91]. Это наиболее "реляционная" из всех СУБД аналогичной стоимости при относительно слабых требованиях к объему доступной памяти и производительности процессора.

Среда системы предоставляет ряд средств, из которых наиболее интересны следующие : автоматизированная генерация и интерактивная корректировка экранных форм, шаблонов отчетов, графического представления числовых данных генератора приложений, символьный отладчик.

Самым же мощным средством Paradox, своего рода "визиткой" системы, является компонента выполнения запросов "Ask", с помощью которой пользователь осуществляет выбор данных и подсчет по формулам с применением статистических функций на одной таблице или нескольких одновременно и определяет структуру результирующей таблицы. Ввод параметров запроса очень прост и нагляден, основан на оригинальном интерфейсе QBE (Query By Example - запросе по примеру) [Мейер 87]. При наличии сервера SQL и компоненты Paradox SQL Link есть возможность обращения к серверу как путем трансляции запросов языка QBE в SQL и обратно, так и непосредственно запросами SQL.

Необходимо упомянуть и такие преимущества Paradox, как возможность отката по ходу ручного редактирования таблиц к начальному состоянию, к состоянию в зафиксированный момент или пошагово, оптимальное управление объектами (своппинг процедур и буферов по мере необходимости), высокая скорость транзакций, возможность

шифрования таблиц и приложений с доступом по паролю, а также аккуратная поддержка наиболее известных сетевых операционных систем и драйверов расширенной памяти.

Для пользователей, не имеющих опыта программирования, имеется встроенный генератор приложений (PProg - Personal Programmer), после "общения" с которым формируются программы на языке PAL (Programming Application Language - язык программирования приложений). Программисты, видимо, предпочтут составлять приложения самостоятельно, к их услугам текстовый редактор программ и удобный символьный отладчик.

Paradox - интерпретирующая система, поэтому подключение разноязыких модулей в рамках среды затруднительно. Для подобных целей имеется пакет Paradox Engine, содержащий файлы настройки и библиотеки функций для компилирующих систем Microsoft C, Borland C, Turbo Pascal и среды Windows. Но, к сожалению, Paradox Engine эмулирует работу лишь нескольких основных нереляционных команд и функций Paradox и не поддерживает разработанные в Paradox экранные формы, отчеты, графы и прочие элементы, что значительно снижает его полезность.

В случае затруднения можно воспользоваться системой помощи (клавишей F1), зависимой от контекста, с выходом на иерархию разделов помощи или на алфавитное представление ключевых слов. Впрочем, подсказки в экранах помощи весьма поверхностны, что свойственно большинству продуктов фирмы Borland. Системная документация и литература [Simpson 90] объемна, точна и содержит все необходимые сведения, если конечно у вас хватит терпения всю ее изучить.

Среди недостатков можно назвать следующие.

1. При просмотре / корректировке таблицы изображение на экране жестко задано и определяется расположением и размером открытых таблиц, если действует режим "табличный вид" (table view), а в режиме "вид формы" (form view) - конструкцией формы, разработанной в генераторе экранных форм. Экранная форма может представлять поля либо одной записью текущей и связанных с ней таблиц, либо несколькими записями текущей таблицы, но не обоими вариантами вместе и только одной формой в текущий момент. Причем команды интерактивной работы, получив управление, обновляют экран целиком. Небольшая лазейка остается за счет двух верхних строк, заполняемых опцией PROMPT с двумя параметрами. Кроме того, нельзя связывать таблицы "по цепочке", а можно "привязать" к одной таблице до девяти зависимых.

2. Нет средств сохранения / восстановления участков экрана.

3. Массивы - только глобальные, которые не могут быть переданы в качестве параметров процедуры.

4. Средства обработки текстовых файлов ограничены выводом информации в файл и импортом / экспортом.

5. При выполнении больших по объему программ задержки на подкачку кода с диска могут развить у квалифицированных программистов чувство собственной неполноценности.

В целом выбор Paradox в качестве инструмента предпочтителен в одном из следующих случаев: 1) вам нужно быстро сделать прототип прикладной системы; 2) принципиальны требования производительности операций с большими таблицами; 3) характер действий пользователя требует сложного языка манипулирования данными.

Вместе с тем эффективность Paradox как инструмента значительно падает при определенных условиях. Первое. Если "стандартный" интерфейс интерактивных команд не устраивает пользователя (например, двустрочное меню, реакция на неверный ввод, выбор из словаря значений с подстановкой в главную таблицу, ввод параметров в режимах запроса (Ask), сортировки, кросс-табуляции и других, представление таблиц на дисплее). Второе. Если объем кода PAL программы значительно превышает объем свободной оперативной памяти (при RAM в 1 Мбайт Paradox резервирует примерно 70 Кбайтов под приложения и столько же - для хранения буферов таблиц). Третье. Если принципиальные с точки зрения быстродействия участки приложения реализованы в элементарных командах обработки отдельных записей, полей, переменных вместо реляционных команд манипулирования таблицами.

В первом случае приходится перепрограммировать на PAL интерфейс или целиком команды (рекомендуется обратить внимание на поставляемую с пакетом компоненту ToolKit), во втором - соглашаться с потерей производительности из-за подкачки процедур, которую к тому же надо обеспечить командами загрузки / выгрузки процедур или созданием "закрываемых" процедур, а в третьем случае подумайте, не лучше ли воспользоваться библиотекой Paradox Engine и работать с языком C.

В качестве частичного решения проблем, касающихся англоязычного интерфейса, в настоящее время предлагаются русификаторы Paradox [Ганелин 91], [Фигурнов 91].

Но с другой стороны: солнце лучше тем, что светит и греет; а месяц только светит, и то лишь в лунную ночь.

Козьма Прутков. "Мысли и афоризмы", 52.

1.2. CLIPPER

Clipper - это одна из инструментальных систем, проверенных временем. За последние 5-6 лет значительное количество приложений в нашей стране разработано именно на Clipper - это различные АРМы, применяемые в бухгалтериях, на автотранспортных предприятиях, на заводах и т.д.

С момента появления Clipper [Стогний 89] использовался как альтернатива широко известному dBase, используя одинаковую с последним структуру файлов, общую парадигму языка и большое количество совпадающих команд и функций, но имел очень важное отличие - формирование приложения в виде .EXE-файла, что позволяет достигать приличной производительности даже на таких ЭВМ, как ЕС 1840 без винчестера и с оперативной памятью 512K.

В чем недостатки Clipper ? Если коротко - его "обгоняют" конкуренты. Из последних достижений "родственных" пакетов можно отметить: в dBase IV - реализацию языка SQL, в FoxPro 2.0 -

выход в лидеры по производительности операций с таблицами [Кэлман 91], и в дополнение к традиционному для него интерпретирующему режиму - формирование EXE модулей. Профессиональные разработчики начинают все больше посматривать на интересный пакет Clarion, не говоря уж о таком монстре, как Paradox.

А по существу можно отметить следующие принципиальные неудобства, замедляющие процесс разработки.

1. Процесс разработки приложений в Clipper состоит из вызова большого количества отдельных программ - редакторов таблиц, отчетов, форм и исходных текстов, компилятора, редактора связей. В итоге отладка программ превращается в очень неудобный процесс: редактирование программы и структур таблиц, компиляция (10 - 30 секунд), связывание (в среднем 2 - 3 минуты), запуск EXE программы. Наткнувшись на ошибку, программа "вылетает", выдав сообщение, к примеру, что "в строке XX процедуры YY сообщение ZZ". Далее, запустив редактор, переместившись к строке XX, мы, к примеру, замечаем элементарную опечатку, вызвавшую ошибку "Type mismatch" (смешение типов) - вместо переменной NNN стоит NMN. Если бы Clipper был реализован в виде среды, подобной Turbo C или Quick C, с близкой к ним скоростью компиляции и связывания, подобные ошибки не вызывали бы раздражения.

2. Язык программирования Clipper состоит в основном из элементарных команд, и реальная программа обычно из нескольких тысяч или даже десятков тысяч строк кода.

3. В отличие от "семейств" для каждой таблицы Paradox при программировании в Clipper необходимо держать в голове все имена индексных файлов и формулы индексов, имена форматных файлов (экранных форм), формул фильтров и т.д..

4. Отладчик только в версии 5.0 стал поддерживать исходный текст программ, а в версии 4.0 - только номера строк.

Однако еще рано говорить, что Clipper должен сойти с дистанции. Он позволяет разрабатывать сложные прикладные системы. В составе языка Clipper - около 100 команд и языковых конструкций и около 150 функций, имеются средства использования баз данных для индивидуальных подсистем помощи, контроля данных при вводе или корректировке. Программы обработки ошибок времени выполнения поставляются в виде исходного текста на языке Clipper, что дает разработчикам дополнительные возможности. В версии 5.0 система отличается еще большей открытостью и развиваемостью за счет добавления конструкций языка Си, наличия препроцессорной обработки, условной трансляции, средств определения новых команд, применения так называемых блоков кода и других средств [Nantucket News 91/92], [Лиф 91], [Белл91], [Сверчков 91], [Миропольский 90].

Вместе с системой поставляются файлы - заголовки для компиляции модулей с языков Си и Ассемблер с последующей компоновкой с Clipper программами.

Различными фирмами разработано большое количество библиотек [Биллиг 90], расширяющих возможности системы; из наиболее известных - Clipper Tools One (порядка 150 функций работы с окнами, последовательным интерфейсом, битовыми строками, диском и т.д.), Halo, ClipGraph и Base Object Graphics (графика), MOUSE_T (работа с "мышью"). Полноэкранный отладчик CLIPRO дополняет недостаточные средства отладки в Clipper 4.0. Существует ряд пакетов, существенно расширяющих возможности системы Clipper по формированию экранных форм (БАРК ПЛЮС), отчетов (TERET), разработки приложений (PILOT, CASE_D) и СУБД (FLINT, UNICUM).

2. ОСНОВНЫЕ ОСОБЕННОСТИ

В данном разделе изложены сведения, влияющие на стратегию разработки приложений.

2.1. PARADOX

В Paradox технология работы с таблицами хорошо продумана и интуитивно понятна. При открытии таблицы на экране формируется ее образ (Image), подобный виду таблицы в Clipper в команде DbEdit. Образ занимает область экрана, равную числу записей таблицы, либо может быть уменьшен командой IMAGE. По мере открытия таблиц образы заполняют экран сверху вниз. Для перехода между образами существуют команды UpImage / DownImage (к предыдущему / следующему образу), а также несколько команд позиционирования в конкретный образ или поле образа.

Каждая таблица может иметь семейство (Family) с одним именем, состоящее из экранных форм (расширения файлов "F", "F1"-"F14"), отчетов ("R", "R1"-"R14"), настроек параметров экранного образа ("SET"), контроля данных ("VAL"), главного индекса ("PX") и вторичных индексов ("X??", "Y??"). Все "члены семейства" подготавливаются интерактивно, а элементарность их вызова позволяет существенно "разгрузить" тексты приложений.

Компоненты Paradox доступны в одной из следующих форм : через меню, с помощью клавишных комбинаций, при выполнении приложения (программы), а также путем выполнения мини-программы (в одну строку). Любую последовательность нажатий клавиш можно запомнить в файле (клавишей Alt-F3 начать и закончить запись), а затем "проиграть" в нужный момент (Alt-F4) или включить в текст приложения (файл называется INSTANT.SC). Кстати, это один из способов разработки и отладки приложений.

Для программирования приложений удобно то, что все действия, вызываемые пользователями нажатиями клавиш, имеют команды - эквиваленты, интуитивно очень понятные. Например: команда HOME дает тот же эффект, что и клавиша HOME в том же контексте (аналогично - UP, DOWN и т.д.). Функциональные клавиши имеют двойное наименование - F1/HELP (вызов помощи), F2/DO_IT! (завершение корректировки), F7/FormKey (переключение в форму и обратно), Alt-F7/InstantReport (вызов стандартного отчета на печать) и т.д. В программе употребляются последние наименования.

Таблица 1. Сравнение основных характеристик

N	Раздел / характеристика	Paradox	Clipper
1	Использование аппаратных средств		
1.1	Минимальный объем оперативной памяти	512 Кбайтов	256 Кбайтов
1.2	Минимальный объем системы на диске для разработки приложений	2 Мбайта	1 Мбайт
1.3	Минимальный объем системы на диске для выполнения приложений	1.5-2 Мбайта	Ядро системы 120-150 Кбайтов
1.4	Возможность разработки и выполнения приложений на компьютерах без винчестера	-	+
1.5	Поддержка расширенной памяти:		
	expanded	до 16 Мбайтов	до 1 Мбайта
	extended	до 2 Мбайтов	-
1.6	Возможность работы в защищенном режиме	+	-
2.	Взаимодействие с прочими системами		
2.1	Экспорт/импорт		+
	ASCII	+	+
	dBase (файлы .DBF)	+	-
	Lotus 1-2-3 (файлы .WK и .WK1)	+	-
	Symphony (файлы .WRK и .WR1)	+	-
	Quattro (файлы .WKQ)	+	-
	Reflex (файлы .RXD)	+	-
	Visicalc (файлы .DIF)	+	-
	pfs:File (файлы .PFS)	+	-
2.2	Запуск внутренних команд ДОС		
2.3	Запуск внешних программ	+	
	с загрузкой в свободную область		
	с максимальным освобождением памяти	без 40 K	
2.4	Компоновка приложений с библиотеками (.LIB) и объектными файлами (.OBJ)		
3	Таблицы (базы) данных: ограничения		
3.1	Записей в таблице	2 миллиарда	1 млрд (5.0)
3.2	Полей в таблице	255	1 024
3.3	Размер текстового поля, в байтах	255	64 536
3.4	Число символов в имени поля	20	12
4.	Переменные : ограничения		
4.1	Число переменных в программе	неограниченно	2 048
4.2	Число элементов массива	15 000	4 096
4.3	Число символов в имени переменной	132	12

Таблица 2. Стоимость компонент Paradox

Наименование компоненты	Стоимость в руб.	Стоимость в $ USA	Назначение
Paradox 3.5, русифицированный	19 900	795	Основной пакет
Runtime 3.5	2 600	49.5	Для распространения приложений
LanPack на 1-го пользователя	200		Сетевая поддержка 1-го пользователя
LanPack на 5 пользователей	36 000	995	Сетевая поддержка 5 пользователей
LanPack на 10 пользователей	51 500		Сетевая поддержка 10 пользователей
SQL Link 1.0	8 000	495	Для связи с SQL сервером
Engine 2.0	7 800	495	Библиотека С функций

Таблица 3. Стоимость компонент Clipper

Наименование компоненты	Стоимость в руб.	Стоимость в $ USA	Примечание
Clipper 5.0, русифицированный	30 000	995	Скидка 25% владельцам версии 4.0
Clipper 4.0 и Clipper Tools One, русифицированный	10 000	395	
Clipper Tools One, for 5.0, русифицированный	9 000	350	

Многие команды формируют таблицы, имя которых адекватно характеру действий, например: ANSWER - результат запроса (Ask/Query), STRUCTURE - структура таблицы, LIST - список таблиц, процедур, файлов, пользователей и т.д., KEYVIOL - записи с совпавшим индексным ключом, CHANGED - замененные записи, DELETED - удаленные записи, ENTRY - ввод записей, PROBLEMS - записи, для которых при изменении структуры таблицы значения отдельных полей не могут быть преобразованы. Таблицы-результаты формируются в личном каталоге и как правило открываются автоматически в нижней части рабочего пространства и могут быть использованы как обычные таблицы. Время жизни этих таблиц - до выхода из Paradox. При входе в систему текущий каталог становится личным, а переустанавливается командой SetPrivDir.

В ходе редактирования одной или нескольких таблиц результаты действий - удаление, вставка и модификация полей - записываются в промежуточную таблицу. До момента завершения корректировки имеется возможность пошагового отката (восстановления предыдущего состояния), в момент завершения требуется подтвердить изменения (DO_IT!), а при монопольной корректировке (EditKey вместо CoEditKey) - вернуть исходное состояние. Кроме того, реляционные операции удаления, ввода и модификации формируют таблицы (DELETED, INSERTED, CHANGED), максимально "оберегая" данные от случайной потери. Все эти средства, вместе взятые, создают дружественную обстановку для обучения и экспериментирования.

2.2. CLIPPER

СУБД Clipper не отличается особой сложностью системных принципов.

Каждая таблица располагается в отдельной области, имеющей номер и альтернативное имя, по умолчанию совпадающее с именем таблицы.

Для каждой таблицы могут быть открыты индексные файлы, установлено фильтрующее выражение для управления видимостью записей. Одна таблица может быть связана с несколькими зависимыми, возможна также связь таблиц "по цепочке".

Большинство команд работает относительно текущей записи. В качестве опций команды могут содержать область действия (текущую запись, все записи, несколько следующих, все до конца), а также одно из двух условий продвижения по таблице, заданных логическим выражением : не прекращать (FOR - условие) и выйти, если условие нарушено (WHILE - условие).

На действие большинства команд и функций влияют установки системных переменных (SET - команды).

Для создания и ручной корректировки таблиц и индексов предназначена утилита DBU.exe, для подготовки файлов экранных меток и отчетов существует утилита RL.exe. Последняя применяется редко ввиду ограниченных возможностей, а первая необходима для первоначального создания баз данных.

Обычно вся работа приложения реализуется программой, написанной на языке Clipper, "пропущенной" через компилятор и соединенной редактором связей с прочими компонентами. Существенную часть результирующего EXE модуля составляет ядро системы, примерно 120-160 Кбайтов в зависимости от числа используемых функций.

В следующем номере " Монитора" читайте две главы: представление данных и техника работы с таблицами. В заключительном номере - технология разработки приложений и примеры программ. ■

ЛИТЕРАТУРА

PARADOX

[Браун 91] Эрик Браун. СУБД Paradox наращивает темпы. Мир ПК, № 1, 1991, стр. 113-115.

[Ганелин 91] Ганелин Л.Д. Paradox 3 - лидер, но не в СССР. Серия Библиотека информационной технологии Вып. 2 /Под ред. Г.Р. Громова. - М.: ИнфоАрт, 1991, стр. 108-118.

[Кэлман 91] Дэвид Кэлман. 15 реляционных баз данных: простота доступа, мощные программные средства. PC Magazine/USSR, №2, 1991, стр. 77-92.

[Мейер 87] Д.Мейер. Теория реляционных баз данных. Пер. с англ., М., Мир, 1987, - 608 с.

[Мир ПК 4/91] СУБД Paradox 3.0 и 3.5 (Справочная информация). Мир ПК,№ 4, 1991, стр. 92-93.

[Фигурнов 91] В.Э. Фигурнов. Paradox говорит по-русски. Журнал д-ра Добба. №3 1991, стр. 19-21.

[Хан-Магометов 91] Д. Хан-Магометов, К. Раннев, А. Зотов. Парадоксален ли Paradox?. Компьютер Пресс. №11, 1991, стр. 58-62.

[Чепел 89] Хол Чепел, Мерлайни Найсери. Особенности работы с базами данных в сети: обозрение сетевых характеристик СУБД. Компьютер Пресс. Сборник №1, стр. 51-61.

[Simpson 90] Alan Simpson. Mastering Paradox 3.5. Fifth Edition. SYBEX Inc., 1990, 702 p.

CLIPPER

[Белл 91] Эд Белл. Что такое препроцессор Clipper 5.0 ?. Компьютер Пресс. №10, 1991 г, стр. 47-53.

[Биллиг 90] И.Биллиг. Любителям и профессионалам СУБД Clipper. Компьютер Пресс. №7, 1990, стр. 37-38, 40-41.

[Лиф 91] Г. Лиф. Блоки программного кода. Журнал д-ра Добба. №3 1991 , стр. 22-24.

[Миропольский 90] В.Миропольский. СУБД Clipper 5.0 фирмы Nantucket. Компьютер Пресс. №4, 1990, стр. 10-11.

[Сверчков 91] А.П. Сверчков. Clipper 5.0 - новая система программирования. Компьютер Пресс. №7, 1991, стр. 51-54.

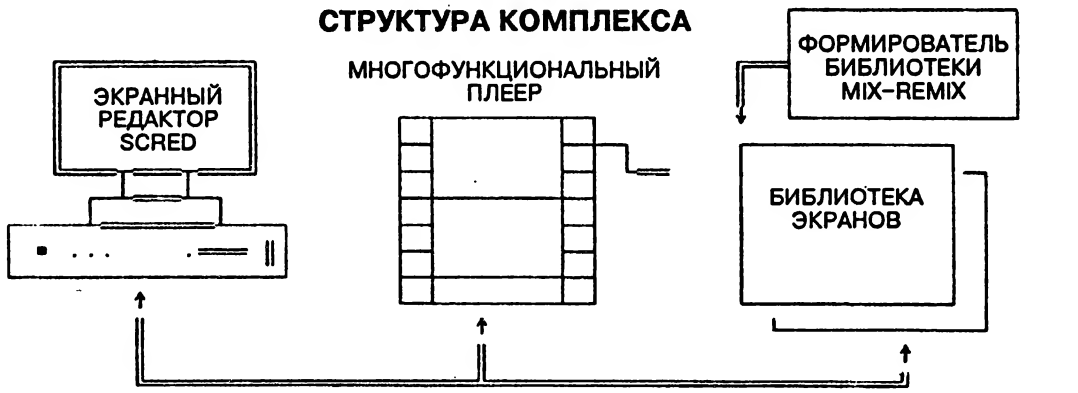
[Стогний 89] Программное обеспечение персональных ЭВМ / Стогний А.А., Ананьевский С.А., Барсук Я.И. и др.; Под ред. Стогния А.А. -Киев: Наук.думка, 1989, стр 8-207.

[Nantucket News 91/92] Nantucket News /Технический журнал фирмы Nantucket. Пер. с англ. /1991-1992 гг.

БЫСТРО, ЭФФЕКТИВНО, КАЧЕСТВЕННО

позволит работать Вам инструментальный программный комплекс фирмы "ИНФОРКОМ"

DESIGN SHOW



ВАША ФАНТАЗИЯ + DESIGN SHOW = ВЫСОКОЕ КАЧЕСТВО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

DESIGN SHOW – позволяет максимально **быстро** создавать экранные формы и фирменные заставки для программ, учебные и демонстрационные слайд–фильмы, используя символы псевдографики

DESIGN SHOW – отличает **простота** в работе

DESIGN SHOW – дает возможность **легко** манипулировать цветами и элементами изображения

DESIGN SHOW – включает 100 оригинальных идей заставок для Ваших программ

DESIGN SHOW – обеспечивает совместимость с: Pascal, Clipper, dBASE, Foxbase

Оборудование: IBM PC XT/AT / EGA, VGA, SVGA /
Поставка на двух дискетах 5. 25, MS DOS 360K.
Цена комплекса–3500 руб. (включая НДС)

НАШ АДРЕС:
121019, Москва, Г–19, а/я 16
Телефон: 452–31–03 (вторник, среда, пятница)
Р/с 500461778 во Фрунзенском комбанке г. Москвы МФО 201412

Компилятор пишется так...

М. Черкашин

У каждой программы может быть свой входной язык. И с такого языка хочется написать компилятор попроще. Существует ряд практических рекомендаций, как это сделать.

Писать компилятор приходится чаще, чем обычно думают. Практически всякая большая система включает в себя входной язык - от примитивного до весьма сложного. Вспомним хотя бы dBASE - это ведь не язык программирования, а система баз данных. На нее даже программы пишутся. А раз нужен входной язык, то бывает нужен и компилятор. И часто нужен быстро. Конечно, всякий предпочтет компактный, быстрый, хорошо оптимизирующий компилятор, но далеко не всякому понравится такой компилятор писать.

Чтобы писать сложные, эффективные и быстрые компиляторы, есть много рецептов. Но здесь речь пойдет не о них. Как написать компилятор просто - вот в чем вопрос! Ведь такой компилятор и короче, и отлаживается легче. Здесь не будет каких-то хитрых алгоритмов, позволяющих достигать чудес эффективности или каких- то особенных способов организации данных. Увы, с этими способами одна проблема: заставить их работать может только чудо. А в этой статье - только практические рекомендации, а работают они надежно!

Оптимизацию и генерацию кода мы рассматривать не будем. Оптимизация - тема для отдельной статьи, а генерация кода - занятие для самоубийцы. Соломоново решение в данном случае - переводить программу не в коды машины, а на язык, который у вас уже есть. Так и делаем. Время компиляции, конечно, увеличивается, но зато мы избавляемся от большого количества неприятной возни.

Модельный компилятор

Лучший способ понять, как писать компилятор - написать его самому. Лучший способ объяснить это - предъявить текст программы. Вот мы и предлагаем статью-программу. Впрочем, ее можно читать и не заглядывая в листинг, но при этом вы много потеряете. Компилятор написан на Паскале, выходной язык - Паскаль, описание входного языка приведено на рис.1. Читатели, владеющие языком Паскаль, узнают и во входном языке знакомые конструкции.

```
Программа на языке SIMPLE есть:
    описания переменных,
    общих для всей программы;
procedure имя;
begin
    описание переменных процедуры;
    операторы;
end;
....
procedure имя;
....

Одна из процедур носит имя main.
Эта процедура и есть программа.

Описание

int список переменных через запятую;
record описание;
....
    описание;
    end   список переменных через запятую;

Присваивание: Имя := [....];
Вызов процедуры: Имя;
Печать: Print имя;
Условный: if [....] then операторы else операторы close;
Цикла: while [....] do операторы close;
```

Рис 1. Краткий обзор языка SIMPLE

Наш модельный компилятор состоит из четырех частей: сканера, блока таблиц имен, основной части компилятора и семантических подпрограмм. В том или ином виде эти части можно встретить в любом компиляторе. Сканер - это глаза компилятора, кото-

рые обращаются к тексту программы только через него. Сканер читает входной файл и избавляет остальные части компилятора от необходимости следить за каждым входным символом. В таблице имен хранится информация о переменных программы. Обращение к ней осуществляется в основном через процедуры блока таблицы имен.

И наконец, основная часть компилятора ...компилирует. Точнее следит за ходом компиляции. Как только становится ясно, с какой конструкцией мы имеем дело, управление передается специализированной, имеющей дело только с этой конструкцией, процедуре, которая называется семантической подпрограммой.

Сканер

Среди всех конструкций, воспринимаемых компилятором, есть минимальные, которые уже не имеет смысла делить на части. Например: идентификатор, число, строка, ключевое слово. Они называются лексемами. Компилятору нет нужды разбираться в их структуре. Кстати в описании старого доброго Алгола-60 использовался термин

```
(LexType = LexKey)

procedure then
main      else
end       close
int       while
record    do
 if       print

 Имена
(LexType = LexNames)

     Например:
sight
gh218
slow1a6

 Числа
(LexType = LexNumber)

     Например:
17
0218
33
1

 Спецсимволы
(LexType = LexSpecs)

:=
,
:

"Нечто в []"
(LexType = LexExpr)

Например:
[a*(8+d)]
[d mod h + 12]
```

Рис 2. Ключевые слова

Пример 1

Сканер

```
(*************************************)
(*   Данные для сканера              *)
(*************************************)
const ExprLen = 240;
Type LexEnum = (LexKeyword, LexNames, LexNumber,
                LexSpecs, LexExpr);
    SpecEnum = (SemiColn, Comma, Assign);
var LexType : LexEnum;
LexVal : integer;
    (* Буфер для выражения    в квадратных скобках *)
Buf : string [ExprLen];
EndStream, LogEOF : boolean;
(* Буфер и указатель буфера
  строки входного файла*)
LineBuf : string [81];
LinePtr : integer;
    (* А вот здесь мы храним символ,
    от которого мы отказались
    с помощью функции UngetCh *)
OldCh : char;
    (* True, если в OldCh что-то лежит*)
 Suspend : boolean;

procedure IniScan;
begin
    LineBuf := ' ';
    LinePtr := 1;
    Suspend :=False;
end;

(*
    Сканер. Оставляет в переменных
        LexType и LexVal.
    Нечто в квадратных скобках
        остается в буфере Buf.
    Функции Alpha (c) и  Num (c)
        равны True, если их аргумент
        буква или цифра соответственно.
*)
procedure Scan;
var Ch : char;
begin
    if Ch = ' ' then       (* Бог с Ним! *)
     else UngetCh;
    if EndStream then LogEOF := true  (* Конец файла! *)
     (* Встретили квадратную скобку - впереди выражение *)
    else if Ch = '['           then Brackets
     (* Впереди ключевое слово или переменная,
     а вот что неясно *)
    else if Alpha (Ch)         then KeyOrNames
    else if Num  (Ch)          then Numbers    (* Число!*)
    else if Ch = ';'           then Spech (SemiColn)
    else if Ch = ','           then Spech (Comma)
    else if Ch = ':'           then begin
        Ch := GetCh;
        Ch := GetCh;
        if Ch = '=' then Spech (Assign); (* := *)
    end;
end;

(*
    Сначала прочитать нечто алфавитно-цифровое,
```

Пример 1. Продолжение

```
      а потом уже разобраться, что это -
      ключевое слово или идентификатор *)

   procedure KeyOrNames;
   var LexBuf : bloc;
       I,t : Integer;
       Ch : char;
   begin
       Ch := GetCh;
       I := 1;
       repeat
       LexBuf [i] := Ch;
       I := I+1;
       Ch := GetCh;
     until not (Alpha (Ch) or Num (Ch));
   LexBuf [0] := chr (I);
   UngetCh;
   t := Find (LexBuf);
   if KeyEntry (t) then begin
       LexType := LexKeyword;
       LexVal  := t;
   end else if t <> 0 then begin
       LexType := LexNames;
       Lexal := t;
   end else begin
       LexVal := LexNames;
       t := PutIn (LexBuf);
   end;
end;

   (* Считать число *)
   procedure Numbers;
           Count : Integer;
           Ch : char;
   begin
           Ch := GetCh;
           Count := 0;
           repeat
               Count := 10*Count + (ord (Ch)-ord ('0'));
               Ch := GetCh;
           until not Num (Ch);
       UngetCh;
       LexType := LexNumber;
       LexVal := Counter;
   end;

   (* Установить переменные LexType и LexVal
       для спецсимволов *)
   procedure Spech (Sym : EnumSpec);
   begin
       LexType := LexSpecs;
       LexVal  := ord (Sym);
   end;

   ( Нечто в квадратных скобках **)
   procedure Brackets;
   var  Ch : char;
        I  : Integer;
   begin
       Ch := GetCh;
       Ch := GetCh;
       i := 1;
       while Ch<>  ']' do
       begin
           Buf [i] := Ch;
           Ch := GetCh;
           i := I+1;
           end;
           Buf [0] := chr (i-1);
           LexType := LexExpr;
   end;

   (* Вспомогательные подпрограммы для сканера *)
   Function GetCh : char;
   var i        : Integer;
       Ch       : char;
   begin
       If Suspend then begin
           GetCh := OldCh;
           Suspend := False;
       end
   else repeat
       if LinePtrlegth (LineBuf) begin
           Ch := ' ';
           NewLine;
       end else begin
           Ch := LineBuf [LinePtr];
           LinePtr := LinePtr+1;
       end;
     until ((Ch<>' ') and (Ch<>chr (Tab))) or Eof (SRC);
    if Eof (SRC) then EndStream := 1;
   end;

   procedure UngetCh;
   begin
       Suspend :=True;
   end;

   procedure NewLine;
   begin
       ReadLn (LineBuf);
       if Eof (SRC) then LinePtr := 1;
       LinePtr := 1;
          end;
```

■

"иероглифы", то есть нечто такое, что можно обозначить значком. Но у компьютера всего-то 256 символов - и приходится писать, например "procedure". А посему идея следующая: отдельная процедура считывает программу из файла, выделяет оттуда лексемы и передает их основной части компилятора. А той до текста программы уже нет никакого дела. Такая подпрограмма называется сканером.

В языке SIMPLE существуют пять видов лексем (рис.2): ключевые слова, идентификаторы, числа, спецсимволы и нечто в квадратных скобках. По поводу первых четырех типов, кажется, все ясно. А пятый, последний - это наша хитрость. Дело в том, что компиляция выражений - это вопрос особый, и ему будет посвящена следующая статья. Поэтому, пользуясь тем, что выходной язык у нас Паскаль, мы перекладываем эту задачу на плечи компилятора с Паскаля. А чтобы у нас не возникало проблем - информацию, которую мы просто переписываем в выходной файл, не разбираясь в его внутренней структуре, заключаем в квадратные скобки.

Посмотрим на текст программы (пример 1.). Основная процедура сканера - Scan. Вызывая ее, мы получаем очередную лексему. Мы можем даже считать ее аналогом процедуры ввода - со странного устройства, которое передает внутрь компьютера лексемы. Процедура Scan по первому символу лексемы определяет, что за лексема нас сейчас ждет, и вызывает "специализированную" процедуру, которая только этим типом лексем и занимается. Единственная загвоздка здесь - ключевые слова путаются с именами. Поэтому они считываются вместе, а уж потом одно отделяется от другого.

Обратим внимание на вспомогательные процедуры GetCh и UngetCh. Процедура GetCh заменяет символы перевода строки и табуляции на пробелы и пропускает лишние пробелы (так получается, что несколько пробелов подряд эквивалентны одному). Процедура UngetCh используется, если мы "погорячились" и считали "лишний" символ. Тогда с помощью этой процедуры можно сказать: "Возьми обратно!" Процедура GetCh выдаст в следующий раз тот же символ.

Таблица имен

Когда человек пишет свою программу или разбирается в чужой, ему нужно держать в голове кое-какую информацию. Но машина железная, у нее головы нет. Поэтому необходима структура данных, которая о каждой переменной хранила бы определенную информацию. Эта структура называется таблицей имен. Без нее не обходится ни один компилятор.

Таблица имен действительно похожа на таблицу. Она состоит из "строк" - ячеек таблицы имен, которые могут содержать информацию об одной простой переменной. Обычно эти ячейки называют записями таблицы имен, но здесь сознательно используется другой термин, чтобы не было путаницы с языком SIMPLE.

Проще всего, когда переменная - целого типа. Для нее нужно хранить всего ничего - имя и тип (integer). В реальных, больших компиляторах, впрочем, есть еще кое-что - ее адрес, например. Но мы, слава Богу, компилируем в Паскаль!

С записями будет посложнее. Даже в голове у нас они хранятся не целиком, а как нечто, сложенное из кусочков. Но в одну ячейку таблицы имен может поместиться информация максимум об одной переменной. Поэтому сейчас наша задача - "раскидать" по ячейкам информацию о структуре записи. Это можно сделать разными способами. Здесь выбран следующий. На любую переменную - i,j,u и т.д. заводим одну ячейку таблицы имен. Если эта переменная - запись, то в поле Fields пишется ссылка на описание структуры ее полей. В данном случае ссылка - просто номер ячейки таблицы имен, содержащей заголовок этого описания.

Описание структуры записи представляет собой заголовок и еще несколько ячеек - по одной на каждое поле. Каждая из этих ячеек (в том числе и заголовок) содержит в поле Ref номер ячейки для следующего поля записи. Ячейка для последнего поля содержит в поле Ref ноль. Это означает "Дальше полей нет!". Если же поле записи снова является записью, то соответствующая этому полю ячейка содержит в поле Fields ссылку на описание структуры полей.

Первые MaxKey записей таблицы имен на самом деле не имена, а ключевые слова. Имена и ключевые слова вообще очень легко перепутать. Поэтому если имя будет найдено в таблице имен, но в записи с номером не большим, чем MaxKey, то это на самом деле не имя, а ключевое слово. Можно конечно завести для ключевых слов отдельный список, но так, как сделано здесь, проще.

Для обращения к таблице имен (пример 2) предусмотрены всего три процедуры: поиск имени в таблице, помещение имени в таблицу и определение по номеру записи, соответствует ли она ключевому слову или нет (это всего лишь сравнение с константой MaxKey - но тем не менее это полезно).

Основной блок компилятора

К моменту, когда вы уже готовы сесть и начать писать текст этого блока, вы должны знать, что у вас делают остальные части (сканер, процедуры работы с таблицами и т.д.). Причем желательно, чтобы остальные части делали побольше, а эта поменьше. Отдельную небольшую подпрограмму вы всегда отладите, а вот процедуру, которая вызывает несколько других процедур, отладить несколько сложнее. Короче, главные проблемы у вас будут все-таки с основной частью.

Посмотрите на текст программы (пример 3). Что должна делать основная часть? Читать описания переменных и заполнять таблицу имен. Потом содержимое таблицы имен должно быть вовремя записано в выходной файл в виде описаний переменных на языке Паскаль. Кроме того, для каждого оператора на языке SIMPLE в выходной файл должен быть записан эквивалентный ему оператор языка Паскаль.

В описаниях основная проблема - это описание записей. "Разобраться" с ними - значит поместить их в таблицу имен. В данной программе это сделано рекурсивно. А действительно, как это еще сделать? Ведь и записи определены рекурсивно: "...содержит в качестве полей целые числа и [другие]_записи".

Есть также определенные проблемы и с трансляцией описаний. Оператор присваивания или печати мы всегда можем считать целиком в память и уже потом компилировать. А вот оператор цикла сам может содержать внутри себя тысячи операторов. Поэтому нам остается только одно - встретив while...do, записать в выходной файл начало оператора цикла, потом как ни в чем ни бывало компилировать внутренние операторы цикла, а встретив close и вспомнив, что когда-то мы встречали while...do, записать в выходной файл все, что останется. К счастью, мы компилируем в Паскаль, а не в коды. И запоминать на самом деле нам ничего не надо. Достаточно, встретив while...do, написать "while ... do begin", а встретив close, написать "end" - и мы никогда не прогадаем, потому что "begin" ставим всегда.

А вот если бы мы транслировали в машинный код, нам необходимо было бы знать, с какого адреса начинался этот оператор. А для этого нужна структура данных, называемая стеком.

Пример 2.

Таблица имен

```
(*************************************************)
(*  Процедуры для таблицы имен           *)
(*************************************************)
const WordLen = 32;
    TabSize = 1024;
    MaxKey = 12;
Type bloc = string [WordLen];
    EnumTag = (Dunno, TagRecord, TagInt);
        (* Ячейка таблицы имен *)
    ncell = record
        Pname : bloc;
        Tag : EnumTag;
        Ref : integer;
        Fields : integer;
    end;
    (* Таблица имен *)
var TableOfNames : array [1..TabSize] of bloc;
    TabLen : integer; (* Размер таблицы имен *)

procedure IniTab;
begin
    TabLen := 0;
    PutIn("procedure");
    PutIn("main");
    PutIn("end");
    PutIn("int");
    PutIn("record");
    PutIn("if");
    PutIn("then");
    PutIn("else");
    PutIn("close");
    PutIn("while");
    PutIn("do");
    PutIn("print");
end;

(* Найти переменную с данным именем *)
int FInd (Name: bloc);
var I : integer;
begin
    FInd := 0;
    for I:=MaxKey+1 to TabLen do
        If Name = TableOfNames [I].Pname;
            then FInd := I;
end;

(* Создать новую запись таблицы имен с данным именем *)
procedure PutIn (Name:bloc);
begin
    TabLen := TabLen +1;
    With TableOfNames [TabLen] do
    begin
        Pname := Name;
        Tag    := Dunno;
    end;
end;

(* Если номер записи не больше MaxKey,
то это на самом деле не имя, а ключевое слово *)
Function KeyEntry (t:integer) : boolean;
begin
    KeyEntry := (t>0) and (t<=MaxKey);
end;
```

Пример 3

Основной блок компилятора

```
Type EnumKey =  (KeyProc, KeyMain, KeyEnd, KeyInt,
                 KeyRecord, KeyIf, KeyThen, KeyElse,
                 KeyClose, KeyWhile, KeyDo, KeyPrint);
var CommonTop : integer;
procedure Compile;
begin
    Scan;
    (* Сначала считаем описания переменных,
    доступных всей программе
    *)
    while LexType <> LexKey
        or LexVal <> KeyProc do
            Declare;
    ProgInit; (* Вывести стандартный
            заголовок программы на Паскале *)
    CommonTop := TabLen;
    while not LogEOF do
        ThisProc; (* Одна процедура *)
    ProgClose;  (* Вывести стандартное
            окончание программы на Паскале *)
end;

(* Скомпилировать одну процедуру *)
procedure ThisProc;
begin
    TabLen := ThisProc;
    while (LexType = LexKey) and
    (LexVal = ord (KeyInt)
        or LexVal = ord (KeyRecord)) do
        Declare;
    ProcInit; (* Заголовок процедуры *)
    Code;
    ProcClose; (* Окончание процедуры *)
end;

(* Обработать одно описание *)
procedure Declare;
var Y:integer;
begin
    if (LexType=LexKey)
        and (LexVal=ord (KeyInt)
    then NameList (TagInt,0)
    else if (LexType=LexKey)
        and (LexVal=ord (LexRecord))
    then begin
            PutIn ('');
            TableOfNames [TabLen].Fields := 0;
            Y := TabLen;
            ReadFields (Y);
            NameList (TagRecord,Y);
        end;
 end;

(* Считать список имен переменных после описателя
    и поместить эти имена в таблицу имен*)
procedure NameList (TagVal, FldVal : integer);
begin
    repeat
        Scan;
        With TableOfNames [LexVal] do
            begin
                Tag := TagVal;
```

Пример 3. Продолжение

```
                Fields := FldVal;
                Ref := 0;
            end;
        Scan;
    until (LexType=LexSpecs)
        and (LexVal=ord (SemiColn));
end;

(* Обработать поля записи *)
procedure ReadFields (Y0 : Integer);
var X,Y,Pre : Integer;
begin
    Y := Y0;
    repeat
        Pre := Y;
        Dcl1 (X,Y);
        TableOfNames [Pre].Ref := X;
    until (LexType=LexKey)
        and (LexVal=ord (KeyEnd));
        Table OfNames [Y].Ref := 0;
end;

(* Dcl1 делает почти то же, что и Declare,
    только описание, которое он считывает,
    является полем записи.
    Поэтому вместо NameList используется
    другая процедура
*)
procedure Dcl1 (var X,Y : Integer);
var C:integer;
begin
    if (LexType=LexKey)
        and (LexVal=ord (KeyInt))
    then NameList1 (TagInt,0)
    else if (LexType=LexKey)
        and (LexVal=ord (LexRecord))
    then begin
            PutIn ('');
            TableOfNames [TabLen].Fields := 0;
            C := TabLen;
            ReadFields (C);
            NameList1 (TagRecord,C,X,Y);
        end;
end;

(*  Разница между NameList и NameList1 в том,
    что последняя устанавливает "указатели" Ref. *)
procedure NameList1 (TagVal,FldVal : Integer;
                                var X,Y : Integer);
begin
    X:=0;
    Pre:=0;
    repeat
        Scan;
        if X=0 then X:=LexVal;
        Y:=LexVal;
        With TableOfNames [LexVal] do
            begin
                Tag :=TagVal;
                Fields :=FldVal;
            end;
        If Pre0 then
            TableOfNames [Pre].Ref := LexVal;
        Pre:=LexVal;
        Scan;
    until (LexType=LexSpecs)
        and (LexVal=ord (SemiColn));
end;

(* Скомпилировать все операторы в процедуре *)
procedure Code;
begin
    repeat
        Statement;
    until (LexType=LexKey)
        and (LexVal=ord (KeyEnd));
end;

(* Скомпилировать один оператор *)
procedure Statement;
begin
    if LexType = LexKey then
        (*Большинство операторов можно распознать
            по первому ключевому слову*)
    begin
        if LexVal = ord (KeyIf)
            then SempIf
        else if LexVal = ord (KeyElse)
            then SempElse
        else if LexVal = ord (KeyWhile)
            then SempWhile
        else if LexVal = ord (KeyClose)
            then SempClose
        else if LexVal = ord (KeyPrint)
            then SempPrint
    end
        else if (LexType=LexNames)
        then SempAssignOrCall;
(* Присваивание или вызов процедуры *)
 end;

(* Семантические подпрограммы *)

(* if...then *)
procedure SempIf;
begin
    WriteLn ('if');
    Scan;
    WriteLn (Buf);
    WriteLn ('then begin');
    Scan; (**)
    Scan;
end;

(* Else *)
procedure SempElse;
begin
    WriteLn ('end else begin');
    Scan;
end;

(* Print...*)
procedure SempPrint;
begin
    Scan;
    Write ('WriteLn');
    id := TableOfNames [LexVal].Pname;
    Write ('(',''',id,' = ',''',',',id,')');
```

Пример 3. Продолжение

```
Scan;  (*;*)
    Scan;
  end;

(* По первой лексеме нельзя отличить
    вызов процедуры от присваивания.
    Поэтому отличаем одно от другого
    в одной семантической подпрограмме
    по ходу дела
*)
procedure SempAssignOrCall;
begin
    Write (TableOfNames [LexVal].Pname);
    Scan; (*:=*)
    if (LexType = LexSpecs) and (LexVal = ord (Assign))
        (* Присваивание *)
    then begin Scan;
            WriteLn (': = ',Buf,';');
            Scan; (*;*)
        end
    else if (LexType = LexSpecs) and (LExVal = ord (SemiColn))
        then  WriteLn (';'); (*  Вызов процедуры *)
        Scan;
end;

(* While...do *)
procedure SempWhile;
begin
    WriteLn ('While');
    Scan;
    WriteLn (Buf);
    WriteLn ('do begin');
    Scan;
    Scan;
end;

(* Close *)
procedure SempClose;
begin
    WriteLn ('end;');
    Scan;
 end;
```

Куда идти дальше?

У тех, кто дошел до этого места, возможно возникнут вопросы. Я не берусь предугадать их все, но на некоторые можно ответить заранее.

- Что еще нужно сделать, если с компилятором будут работать часто?

- Необходимо вставить диагностику ошибок. На практике компилятор в большинстве случаев выдает не скомпилированный текст, а сообщение об ошибке. Поэтому и писать компилятор нужно так, чтобы проверка на ошибки производилась как можно легче.

В частности, чем больше хитрых приемов вы используете (например ускоряющих компиляцию), тем больше вам придется ломать голову над тем, куда этот алгоритм забредет в случае ошибки и как сделать, чтобы копилятор не сбился с пути и обнаружил не только первую встретившуюся ошибку, но еще и какие-то другие.

-Как вставлять новые конструкции в язык?

-Если речь идет о структурах данных, то лучше добавить новое поле в запись таблицы имен, чем ломать голову над тем, как сэкономить лишнее поле.

Если речь идет об операторах - добавьте еще одну специализированную процедуру, компилирующую данный оператор. Соответственно вызов этой процедуры должен быть вставлен в процедуру Code.

-А если хочется все-таки сделать компилятор более эффективным?

-Ради Бога! Только сначала пусть он заработает. А потом его можно оптимизировать по вкусу. Кроме того, прежде чем применить многообещающий алгоритм, хорошенько проверьте, будет ли он столь эффективен в вашем случае. А то ведь все эти алгоритмы на практике хороши в весьма специальных случаях. Подумайте, стоит ли овчинка выделки.

-Как отлаживать компилятор?

-Компилятор - очень коварная программа. И выловить все ошибки из него обычно не удается. Не полагайтесь на свою интуицию. Сделайте себе набор тестов - небольших, но сам набор пусть будет представительным. И когда компилятор будет подходить к готовности, прогоните их все. (Да, в том числе и те, что уже когда-то проходили!)

И наконец, еще один совет. Больше простоты! Если бы программисты всегда ему следовали, то работающих программ было бы несомненно больше (а недоотлаженных, брошенных на полдороге - меньше). ■

Российское научно-техническое общество РЭС им. А. С. Попова
Министерство связи России
приглашают фирмы и организации принять участие в
3-й Международной выставке-ярмарке программных и аппаратных средств

13-17 октября 1992, Москва Всероссийский выставочный центр (бывшая ВДНХ), павильон 4

Осенние выставки- ярмарки *Softool* - это
уникальная возможность реализации Ваших разработок!
Выставку проводит МП ЭКСПОСЕРВИС
Заявки на участие принимаются по адресу: 103897, Москва, Кузнецкий мост, 20/6,
ЦП РНТО РЭС им. А.С.Попова, МП ЭКСПОСЕРВИС
Телефон: 921-06-59. Тел./факс: 924-70-72.

Программный интерфейс между системой AutoCAD и форматом CLDATA

А.Крапивенко

Файлы обмена чертежами системы AutoCAD

Система AutoCAD может использоваться сама по себе, как полный редактор чертежей. Однако иногда в других программах возникает необходимость анализировать чертежи, созданные системой AutoCAD, или же осуществлять генерацию чертежей с помощью других программ, а их оценку, изменение и вывод на плоттер с помощью системы AutoCAD.

Например, если вы создали архитектурный чертеж с помощью системы AutoCAD, используя "вставные" части для изображения окон, дверей и т.д., то вы можете осуществить обработку файла чертежа и составить смету на материалы по всем элементам, или даже определить потребление энергии, исходя из площади, количества и типа оконных проемов. Другим примером может служить использование системы AutoCAD для описания структур, которые затем передаются на большую вычислительную машину для проведения структурного анализа по методу конечных элементов. Вы можете вычислить напряжения и смещения, а затем передать информацию обратно для изображения деформированной структуры с помощью системы AutoCAD.

Так как база данных чертежей системы AutoCAD хранится в очень сжатом формате, то программам пользователя непосредственно прочитать эту информацию трудно. Кроме того, различные машинные реализации системы AutoCAD могут использовать различные внутренние форматы для базы данных, подобранные для получения максимальной производительности вычислительной машины, на которой запущена система AutoCAD. Для обеспечения возможности обмена файлами чертежей между различными машинными реализациями системы AutoCAD, а также между системой AutoCAD и другими программами были определены форматы файла "обмена чертежами".

Файлы DXF обмена чертежами в коде ASCII

Одним из таких форматов является формат файла DXF обмена чертежами. DXF файлы являются стандартными текстовыми файлами ASCII. Они могут быть просто перетранслированы в форматы другой системы проектирования или добавлены к другим программам для специализированного анализа. Автор встречался со многими системами, поддерживающими данный формат. Однако из-за отсутствия в данном формате векторных шрифтов, предпочтение было отдано формату DXB. Подробное описание этого формата содержится в фирменном описании системы ACAD (приложение С).

Двоичные файлы обмена чертежами

Формат файла DXB

Формат файла DXF, описанный ранее, представляет собой полное представление чертежа системы AutoCAD в текстовой форме кода ASCII, допускающей простую обработку с помощью других программ. Однако в случае использования программы CAD/camera или же других программ, обращение к которым осуществляется посредством средств "внешних команд", обеспечена возможность использования гораздо более компактного файлового формата "DXB" (для двоичного обмена чертежами).

К сожалению, данная информация, предоставляемая фирмой, предназначена для опытных программистов и может быть изменена без уведомления. Например, версия 9.0 ACAD содержит расширения в формате файла DXB для обеспечения соответствия примити-

вам 3DLINE и 3DFACE и предоставления возможности изменять цвета отдельно от изменения слоев.

Файл DXB имеет следующий формат.

Заголовок: "AutoCAD DXB 1.0" CR LF ^Z NUL (19 байтов)

Данные: ...Нуль или более записей данных...

Признак конца: NUL (1 байт)

Каждая запись данных начинается одним байтом, который следует перед элементами данных и указывает ее тип. Элементы данных имеют различные формы представления и кодирования данных. В приведенном ниже описании каждый элемент данных имеет префикс в виде буквы и дефиса (табл. 1).

w- 16-битовое целое число, имеющее обратный порядок следования байтов, как в стандартных словах процессора 8086 (сначала младший байт, затем старший).

l- указывает на 32-битовое целое число, которое также имеет обратный порядок следования байтов.

f- перед значением указывает, что это 64-битовое число с плавающей точкой типа IEEE (как для процессора 8087).

n- число, которое может быть либо 16-битовым целым, либо числом с плавающей точкой в зависимости от последних установок элемента данных "режим чисел". По умолчанию режим представления чисел 0, целые со знаком. Если он установлен в 1, все элементы n- будут считаться как числа с плавающей точкой.

u- элемент, представляющий либо 32-битовое целое, либо число с плавающей точкой в зависимости от последней установки режима. Если это 32-битовое целое, то значение масштабируется путем умножения на 65 536, если значение с плавающей точкой, масштабирование не применяется.

a- элемент представляет угол. Если режим представления чисел - целочисленный, 32-битовое целое представляет угол в миллионных долях градуса (диапазон от 0 до 360 000 000). Если это число с плавающей точкой, то оно представляет градусы.

Таблица 1. Типы записей данных в формате DXB

Тип записи	Номер	Содержимое записи	Объем
LINE	1	n-fromx n-fromy n-tox n-toy	9 байтов
POINT	2	n-x n-y	5 байтов
CIRCLE	3	n-ctrx n-ctry n-rad	7 байтов
TRACE	9	n-x1 n-y1 n-x2 n-y2 n-x3 n-y3 n-x4 n-y4	17 байтов
ARC	8	n-ctrx n-ctry n-rad a-starta a-enda	19 байтов
SOLID	11	n-x1 n-y1 n-x2 n-y2 n-x3 n-y3 n-x4 n-y4	17 байтов
POLYLINE	19	n-closureflag	3 байта
VERTEX	20	n-x n-y	5 байтов
3DLINE	2	n-fromx n-fromy n-fromz n-tox n-toy n-toz	13 байтов
3DFACE	22	n-x1 n-y1 n-x1 n-x2 n-y2 n-x2 n-x3 n-y3 n-z3 n-x4 n-y4 n-z4	25 байтов
SCALE FACTOR	12	f-scalefac	9 байтов
NEW LAYER	12	"layername" NUL	strlen("layername")+2
LINE EXTENSION	130	n-tox n-toy	5 байтов
TRACE EXTENSION	131	n-x3 n-y3 n-x4 n-y4	9 байтов
BLOCK BASE	132	n-bx n-by	5 байтов
BULGE	133	u-2h/d	5 байтов
WIDTH	134	n-startw n-endw	5 байтов
NUMBER MODE	135	w-mode	3 байта
NEW COLOR	136	w-colornum	3 байта
3DLINE EXTENSION	137	n-tox n-toy n-toz	7 байтов

Все длины заданы из предположения, что режим чисел установлен в 0 (режим целый). Если режим чисел - вещественный, добавляются 6 байтов к длине каждого n- элемента и добавляются 4 байта для каждого a- или u-элемента.

Элемент LINE EXTENSION /УДЛИНЕНИЕ ЛИНИИ/ позволяет продлить последнюю линию или ее удлинение от ее конечной точки (точки "to") до новой конечной точки ("to point"). Элемент удлинения трассы подобным же образом позволяет продлить последнее трассированное сплошное тело или же удлинение трассы от ее конечной линии с координатами X3, Y3, X4, Y4 до новой линии с координатами X3, Y3, X4, Y4.

SCALE FACTOR /МАСШТАБНЫЙ КОЭФФИЦИЕНТ/ представляет собой число с плавающей запятой, которое при умножении на целочисленные координаты дает возможность получить координаты с плавающей точкой, используемые в реальных графических примитивах. При считывании файла начальный масштабный коэффициент равен 1.0. С помощью элемента NEW LAYER /НОВЫЙ СЛОЙ/ может создаваться слой, если таковой не существует, имеющий те же самые значения по умолчанию, что и команда "LAYER NEW", и этот слой будет принят в качестве текущего для последующих графических примитивов. По завершении загрузки файла слой будет восстановлен перед выполнением команды.

Элемент BLOCK BASE /БАЗА БЛОКА/ задает базовую (начальную) точку создаваемого блока. База блока должна быть определена до появления первой записи графического примитива. Если блок не определен в файле DXB, то данное описание игнорируется.

Ломаная состоит из прямолинейных сегментов фиксированной ширины, соединяющих вершины, исключая те, которые описаны с помощью элементов BULGE /КРИВИЗНА/ и WIDTH /ШИРИНА/. Признак смыкания должен быть равен либо 0, либо 1; если он равен 1, то существует неявный сегмент от последней вершины (непосредственно перед элементом SEQEND) к первой.

Элемент BULGE, расположенный между двумя элементами VERTEX /ВЕРШИНА/ (или же после последнего элемента VERTEX для замкнутой ломаной), указывает, что две вершины соединены не прямолинейным сегментом, а дугой. Если сегмент прямой линии, соединяющей вершины, имеет длину d, а кратчайшее расстояние от середины этого сегмента до дуги -h, то численное значение составляет: (2*h/d). Кривизна имеет отрицательное значение, если при перемещении по дуге из первой вершины ко второй движение осуществляется по часовой стрелке. Таким образом, половина окружности имеет кривизну, равную 1 (или -1). Кроме этого, элементы BULGE могут масштабироваться с коэффициентом 216. Если режим представления чисел установлен в плавающую форму, то обеспечиваемое значение с плавающей точкой составляет лишь 2*h/d (не масштабированное).

Элемент WIDTH указывает начальную и конечную ширину соединяющего две вершины сегмента (прямолинейного или криволинейного). Данная ширина используется до тех пор, пока не встретится следующий элемент, задающий ширину, или же до элемента SEQEND. Если между элементом POLYLINE (ЛОМАНАЯ) и первой вершиной VERTEX стоит элемент WIDTH, то он хранится как элемент, определяющий значение ширины по умолчанию для ломаной; если ломаная имеет несколько сегментов такой ширины, то использование данной возможности даст значительную экономию пространства базы данных.

LINE и 3DLINE разделяют одни и те же ячейки для запоминания последней точки -в, так что вам не следует смешивать группы расширения для этих двух примитивов в отсутствие первоначальной группы до расширения. Для 3DFACE нет группы "расширения", как и нет явного края, от которого начинать расширение.

Группа NEW COLOR указывает цвет для последовательно идущих примитивов в файле DXB. Аргумент слова "w-colornum" находится в диапазоне от 0 до 256. 0 означает установку цвета по блоку, 1-255 - стандартные цвета системы AutoCAD, и 256 означает установку цвета по слою. Цвет за пределами диапазона 0-256 приводит к установке цвета снова в текущий цвет примитива (вы можете делать это намеренно, что является довольно удобным). Первоначальный цвет примитива материала, добавляемого командой DXBIN, является текущим цветом примитива.

Элемент РЕЖИМ ПРЕДСТАВЛЕНИЯ ЧИСЕЛ управляет режимом представления элементов с типом n-, a- или u- в табл.1. Если установленное значение не нуль, эти значения будут целыми, в противном случае - с плавающей точкой. Хранение и явные масштабные преобразования для этих значений в обоих режимах описаны выше.

Примечание. Как было выяснено на практике, ACAD 9.0 и 10.0 при создании DXB "предпочтение" отдает линиям и их удлинениям. Эксперименты с другими версиями не производились.

Programmer

Для обеспечения возможности обмена файлами чертежей между различными машинными реализациями системы AutoCAD, а также между системой AutoCAD и другими программами были определены форматы файла "обмена чертежами"

Запись файлов DXB

Не существует явной команды AutoCAD для записи файла DXB, однако специальный драйвер плоттера ADI может записать такой файл. Если вы хотите создать файл DXB из чертежа AutoCAD, сконфигурируйте ADI плоттер и выберите возможность вывода файла DXB.

Стандарт обмена файлами IGES

Формат файлов IGES и отображение, необходимое для перекодировки между стандартом чертежа AutoCAD и IGES, не описаны в рамках руководства по системе ACAD. Если вам необходимы детали такого сорта, напишите в фирму Autodesk, Inc. для получения последних спецификаций интерфейса AutoCAD/IGES.

Международный формат обмена чертежами CLDATA

Данный формат был создан для описания языка технологического постпроцессора. Он содержит информацию о контурах детали, движении инструмента и других технологических тонкостях. Формат файла CLDATA соответствует стандартам ISO 3592 и ISO 4343. Этот формат обеспечивается двумя видами представления данных: бинарном и текстовом. Версия программы, описанной ниже, предназначена для создания текстового файла, используемого как в зарубежных системах, так и в отечественных разработках (системе фигурного раскроя плоских деталей САПФИР, МАИ, кафедра 606). Каждая строка представляет собой логическую запись, имеющую заголовок следующей структуры: C0 - число слов в записи, C1 - порядковый номер записи , C2 - тип записи (табл. 2), C3 - подтип записи. Элементы с C4 по C61 - параметры записи.

В связи с разной ориентацией систем ACAD и систем, использующих формат CLDATA, представляется возможным создавать в выходном файле только строки с кодом 5 000 (т.е. только описания создаваемых контуров). Строки с командами технологических постпроцессоров, ошибками движения, допусками на поверхность и др. необходимо генерировать уже в оболочке технологических процессоров. Строка с кодом 5 000 в системе САПФИР имеет следующий вид.

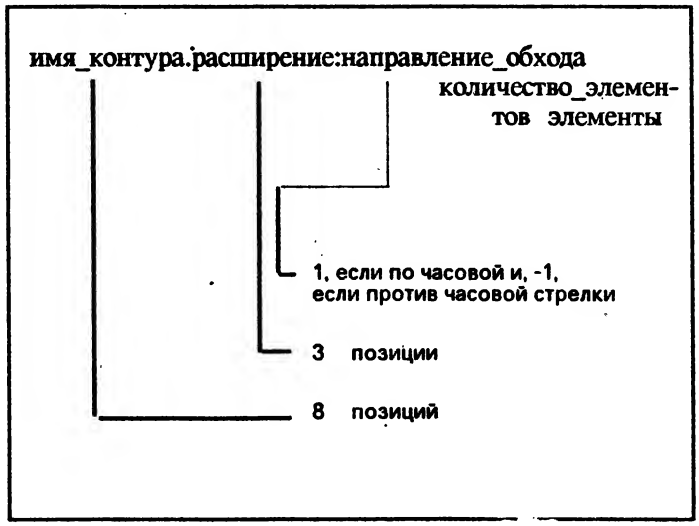
Элементы записаны по формату F10.3 и имеют следующую структуру :

3 - отрезок (удлинение линии) <крайняя точка отрезка>

6 - окружность (дуга окружности) <координаты центра, координаты крайней точки>

(Началом следующего элемента в контуре считается конец предыдущего элемента).

Реализация интерфейса на языке С



Процесс написания программы, реализующей связь с системой AutoCAD посредством алгоритма обработки информации, часто кажется более трудным делом, чем это есть на самом деле.

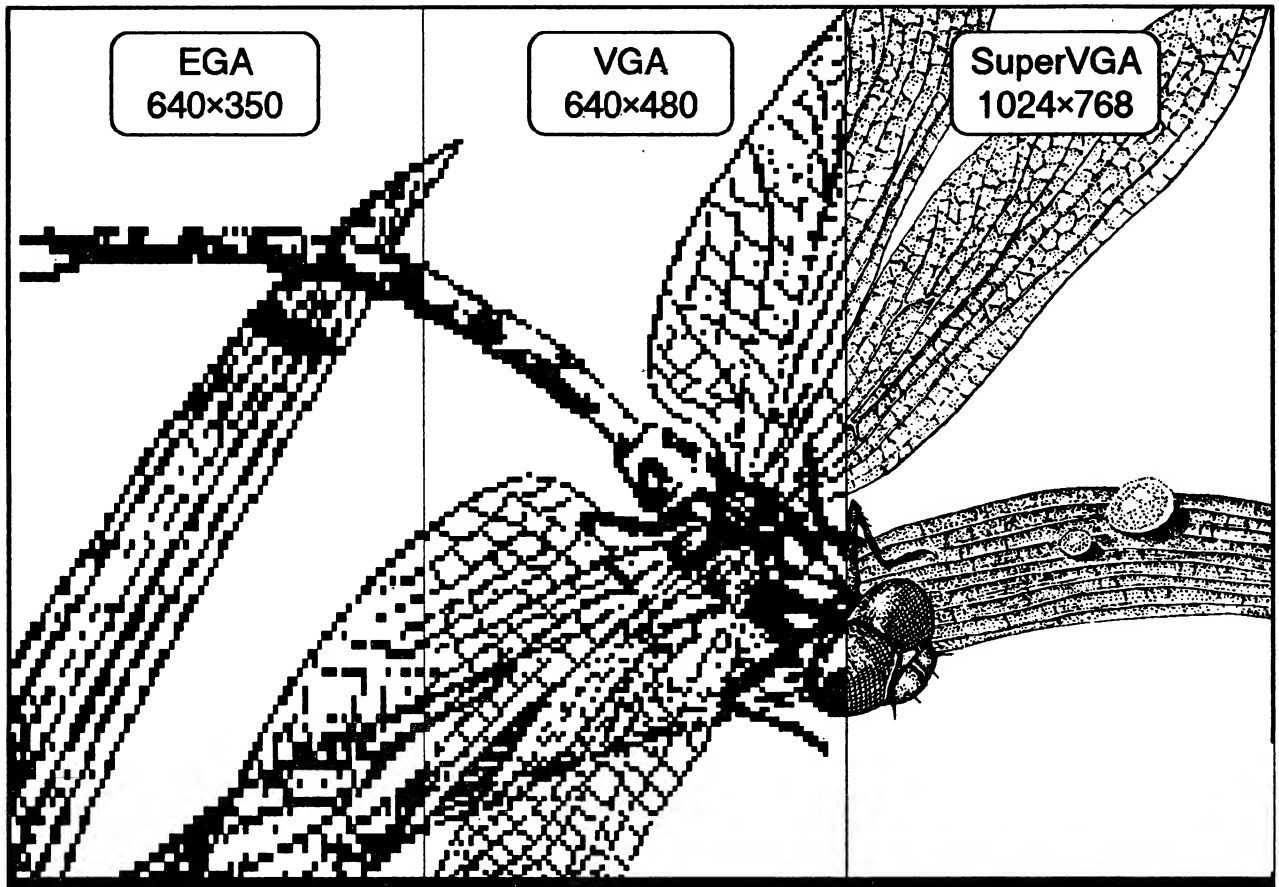
Форматы преднамеренно построены таким образом, что избыточная для Вас информация без особого труда может быть опущена. Структура программы весьма проста (листинг). Однако генерация файла CLDATA имеет свои сложности. Необходимость распознавать границы контуров заставляет сравнивать длину отрезка, соединяющего конечную точку текущего и начальную следующего элементов, с заданной ранее дискретой. При превышении заданного значения происходит создание нового контура. Определение направления обхода контура требует дополнительного алгоритма, рассмотрение которого выходит за пределы данной статьи. ■

Таблица 2. Некоторые типы записей данных в формате CLDATA

Код типа	Содержание	Примечание
1000	Порядковый номер строки	————————
2000	Команда постпроцессора	Код команды содержится в подтипе
5000	Траектория инструмента	Последовательность точек, отрезков и дуг при движении инструмента
6000	Вспомогательные данные	Допуски, диаметр инструмента и фиктивное движение
13000	Ошибка в исходной точке	Задает код ошибки движения программе
28000	Описание контура детали	Записи типов 28 000 и 29 000
29000	Технологическая информация	Используют для управления технологическими процессами

Листинг

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <mem.h>
#include <conio.h>
#define ELEM struct kkk
#define KONTUR struct outp
#define DISCREET 0.1
/*              Перекодировщик чертежей
                ACAD /DXB/ —> CLDATA 1992 г. МАИ СКТБ 606 */
int EndOfCont (float,float,float,float);
void OutputContur ();
void OutputCLDATA ();
ELEM                    /* Информационная часть контура          */
{
int tip;                /* Тип элемента                          */
float ki[5]             /* Координаты элемента                   */
ELEM *sled;             /* Указатель на след. элемент контура    */
};
KONTUR                  /* Формат CLData                         */
{
char *namecont;         /* Имя контура  расширение  */
unsigned kolelem;       /* Количество элементов     */
KONTUR *next;           /* Указатель на след.контур*/
ELEM *BegOfElem;        /* Начало списка элементов */
};
KONTUR *oup,*beg;       /* Один контур              */
ELEM *k2,*m;            /* Содержимое чертежа       */
FILE *f1,*sapfir;       /* Указатели на файлы       */
double ScaleFactor = 1.0,temp[8];     /* Масштабирующий коэфф.*/
float oldx,oldy,  unsigned NumberMode = 2,i,allnomer = 0,nomcont;
int ch = 1,j;
long arga;                      /* Перевод градусов      */
void *buf;
char *kk;                       /* Имя выходного файла   */
int EndOfCont (float x1,float y1,float x2,float y2)
{
if ( fabs( x1 - x2) < DISCREET)
    if ( fabs( y1 - y2) < DISCREET)
        return (0);
return (1);
}
void OutputContur ()            /* Создание нового контура */
{
char *kkk;                      /* Служебная переменная    */
KONTUR *m,   allnomer + = 3;
m = (KONTUR *) malloc (sizeof (KONTUR));
oup -> next = m;
oup = m; /* Резервирование памяти под новый контур*/
k2 -> sled = 0;                 /* Конец списка          */
k2 = oup -> BegOfElem = (ELEM *) malloc (sizeof (ELEM));
oup -> namecont = "";           /* Создание имени контура */
itoa (nomcont+ +, oup -> namecont,2);
strcat ( oup -> namecont, "   . BK"),  oup -> kolelem = 0;
}
void OutputCLDATA () /* Вывод сформированных контуров */
{
ELEM *m,   KONTUR *r, unsigned i,j = 0;
    fprintf ( sapfir, "%2d%5d\n", nomcont, allnomer );
    r = beg;        /* Движение по списку */
do {
        fprintf( sapfir, "%-10s:-1",r -> namecont),
        fprintf( sapfir, "%4d", r -> kolelem+3);
        fprintf( sapfir, "%2d", j+ +);
        m = r -> BegOfElem;   /* Вывод элементов контура */
        do {
            fprintf( sapfir, "   %d.000", m -> tip);
            for (i = 0; i < m -> tip - 1; i+ +)
                fprintf( sapfir, "%10.3f", m -> ki[i]), m = m -> sled;
        } while ( m != 0 );
        fprintf( sapfir, "   0.000\n"),     r = r -> next;
    } while (r != oup);
}
int main ( int argc, char *argv[] )
{
    puts ("\nCopyright (C) 1992 by Andrey.V.Krapivenko\n");
    puts ("Decoder DXB files of ACAD —> files of CLDATA\n");
    if (argc < 2) {
        puts ("Usage: ak.exe name_file.dxb\n");
        return (0);
    }
    kk = (char *) malloc (sizeof (char)*12);
    f1 = fopen (argv[1], "rb" );        /*  Открытие файлов */
    strcpy (kk, argv[1]),  strcpy (strpbrk (kk, ".") + 1, "spr");
    sapfir = fopen (kk, "wt");
fseek ( f1, 19, SEEK_SET ); /* Пропуск информации о версии ACADa */
    beg = oup = (KONTUR *) malloc (sizeof(KONTUR));
                            /* Создание первого контура */
    oup -> BegOfElem = k2 = (ELEM *) malloc (sizeof (ELEM));
    oup -> namecont = "   . "; oup -> kolelem = 0;
    memcpy (oup -> namecont, kk, strlen (strpbrk (kk, ".") - 1));
    strcpy (strpbrk (oup -> namecont, ".") + 1, " HK");
    while ( (ch = fgetc( f1 ) ) > 0 )
    {
    switch ( ch )
    {
        case 1 :
            for ( i = 0; i<4; i+ + ) {
            fread(buf,1,NumberMode,f1)  memcpy (&j,buf,NumberMode);
                temp[i] = j * ScaleFactor;
            }
            if (!oup -> kolelem) {
                oldx = temp[0]; oldy = temp[1];
            }
            else
            if (EndOfCont(oldx,oldy,temp[2],temp[3]) &&
                EndOfCont(oldx,oldy,temp[0],temp[1])) {
                    OutputContur();
            }
            else {
                m = (ELEM *) malloc(sizeof(ELEM));
                k2 -> sled = m;
                k2 = m;
            }
            k2 -> tip = 3;      /* Тип линии - прямая */
            for ( i = 0; i<2; i+ + )
                k2 -> ki[i] = temp[i];
            m = (ELEM *) malloc (sizeof (ELEM));
            k2 -> sled = m,    k2 = m;
            k2 -> tip = 3;      /* Тип линии - прямая */
            for ( i = 2; i<4; i+ + )
                k2 -> ki[i-2] = temp[i];
            oldx = temp[2]; oldy = temp[3]; /* Запоминание
                                              старых к-т*/
            oup -> kolelem + = 6;   allnomer + = 6;
            break;
        case 128 :
            fread (buf, 1, 8, f1), memcpy (&ScaleFactor, buf, 8);
            break;
                case 130 :
            m = (ELEM *) malloc (sizeof (ELEM));
            k2 -> sled = m; k2 = m; k2 -> tip = 3, for ( i = 0; i<2; i+ + ) {
                fread (buf, 1, NumberMode, f1);
                memcpy (&j, buf, NumberMode);
                k2 -> ki[i] = j * ScaleFactor;
            }
            oup -> kolelem + = 3; allnomer + = 3;
            oldx = k2 -> ki[0]; oldy = k2 -> ki[1];
            if ( !EndOfCont (oldx, oldy, oup -> BegOfElem -> ki[0],
                oup -> BegOfElem -> ki[1]))
                OutputContur ();    /* Контур замкнулся */
            break;
        case 135 :
            fread (buf,1,2,f1),  memcpy (&NumberMode,buf,2);
            if (NumberMode)
                NumberMode = 2;
            else
                NumberMode = 4;
            break;
        case 0 :
            break;
        default :
            printf ("\r\nНеизвестный элемент %d",ch);
        } /* Case */
    } /* Цикл */
    if (!oup -> kolelem)
        OutputContur();
    OutputCLDATA (); /* Запись чертежа в формате  CLDATA */
    fclose ( f1),   fclose ( sapfir);
} /* Конец программы */
```



компьютеры САММИТ СИСТЕМС

Сберечь глаза, сидя перед компьютером по 8 часов в день - проблема. Саммит Системс - это **идеальное разрешение** ваших проблем!

- Ваши потребности растут?
  Наш компьютер совершенствуется!
- Надежность под знаком
  Intel, CHIPS, Quantum, Sony
- SuperVGA монитор
  1024x768 (256 цветов)
- 2 года гарантии

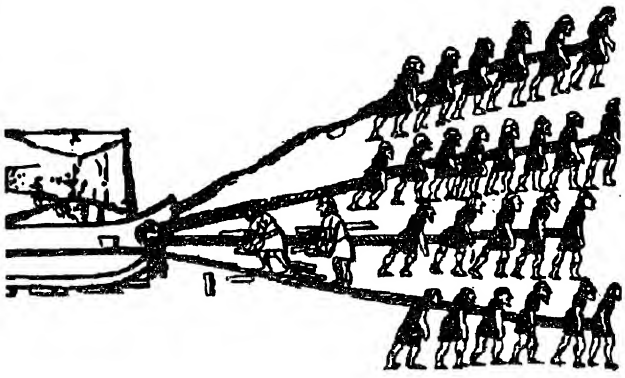
Москва
(095) 299-1162

Минск
(0172) 973-119
973-139
факс 973-519

SUMMIT SYSTEMS

Поставьте Будущее Себе на Стол.

Транспьютеры. Введение в новую концепцию вычислительной техники



М. Дубицкий

Теория ВПП

Как известно всем интересующимся перспективами развития вычислительной техники, некоторые фундаментальные физические законы запрещают повышать быстродействие микропроцессорных элементов до бесконечности. И хотя перспективы для развития еще есть, тем не мение потребности в вычислительных мощностях нарастают, и это приводит к тому, что приходится брать не только скоростью, но и числом процессоров. Все эти причины плюс освоенность и дешевизна производства микросхем в массовых количествах, несомненно приведут к тому, что в скором времени всем нам (вернее, и вам тоже, многоуважаемые программисты) придется осваивать параллельные вычисления, учиться управлять согласованными действиями многих процессоров, работающих над одной задачей одновременно в одном компьютере. И это вовсе не какое-то отдаленное будущее, а реальность завтрашнего дня. Но подробнее об этом ниже, а пока рассмотрим теоретические основы транспьютерных ЭВМ для того, чтобы ясно представить, с чего все начиналось и каковы основы транспьютерной технологии.

Во первых, рассматривая гипотетическую многопроцессорную вычислительную или управляющую систему, мы пока будем считать, что каждый процессор работает под управлением собственной автономной программы, которая хранится в некоторой доступной для него памяти. Хотя во многих современных суперкомпьютерах это не так, но наш случай самый общий и для многих задач нерегулярного характера (особенно для задач управления в реальном времени, но не только) требуется именно такая автономность процессоров.

Теоретически такой способ работы процессоров описывается моделью взаимодействующих последовательных процессов (ВПП или на английском, - CSP (Communicting Sequental Processes)) . Согласно этой модели, программа для многопроцессорного компьютера представляет собой совокупность обычных, полностью автономных, последовательных процессов, которые естественно могут обмениваться результатами своей работы (поскольку обрабатывают одну задачу) по следующему протоколу, который является видимо самой существенной частью модели ВПП. Как только одному из процессов потребуется обменяться порцией информации (вводом или выводом) с некоторым другим определенным процессом, он должен дождаться готовности этого второго процесса к обмену, то есть пока он довыполняется до соответствующих операций приема или выдачи информации с учетом всех условных операторов и всевозможных циклов. Если второй процесс не готов к обмену, то первый кроме ожидания может предпринять какие-то альтернативные действия, но завершить операцию обмена информацией он не может (не только ввода, но и вывода !). Подобная строгость протокола обмена информацией необходима для того, чтобы обеспечить абсолютную уверенность в полной корректности передаваемых сообщений независимо от момента времени, в который происходит обмен (в теории многопроцессорных комплексов эта проблема так и называется - " проблема корректности "). Легко понять, что всякие общедоступные ячейки памяти и буферы конечной длины (а бесконечных не бывает) данному требованию не удовлетворяют ввиду возможного уничтожения информации до того, как она была считана адресатом. И кроме того, подобные средства передачи информации сильно затрудняют понимание и проверку правильности программ. Однако в случае необходимости произвольный буфер в рамках ВПП очень легко организовать с помощью еще одного процесса.

Считается, что весь обмен информацией между процессами происходит по однонаправленным каналам (chanal), которые заранее (до обмена) описаны программистом и определяют способность данных процессов к обмену информацией и направление этого обмена. Как физически организуются каналы между отдельными процессами, мы рассмотрим позже, а пока необходимо отметить следующее. Обмениваясь информацией, оба процесса взаимодействуют не непосредственно друг с другом, а через канал, то есть процесс, посылающий информацию, вводит ее в канал (когда обмен уже разрешен), а процесс получающий воспринимает ее из канала. Если прибегнуть к аналогии, то эта картина очень напоминает ту, когда люди (тоже в некотором смысле последовательные процессы) звонят друг другу по телефону (и даже возможный буфер можно сравнить с автоответчиком). Разница только в том, что по каналу информация передается в строго определенном направлении.

Теоретическое описание и обоснование кратко описанной нами модели ВПП можно найти в книге известного теоретика информатики, профессора Оксфордского университета Чарльза Энтони Ричарда Хоара (C. A. R. Hoare) [1] . Книга интересна тем, что содержит новое теоретико-математическое описание вычислительного процесса, однако не всякий первокурсник с успехом прочтет ее, даже если достанет. Мы же более не станем углубляться в теорию программирования, а перейдем ближе к практике.

Для описания взаимодействующих последовательных процессов группой британских ученых под руководством профессора Хоара был разработан специальный язык программирования Оккам, названный так в честь английского логика и философа XIV века Уильяма Оккама, который, работая в области философских аспектов научных исследований, ввел известный принцип Оккама (называемый также " бритвой Оккама"), суть которого понятным, хотя и очень не точным языком можно выразить так: " не надо выдумывать ничего лишнего ". И действительно трудно найти что-либо лишнее в языке Оккам, хотя он является мощным языком высокого уровня, по стилю чем -то неуловимо напоминающим Паскаль (но с совершенно другими конструкциями). Подробно рассматривать этот язык мы будем в следующих номерах журнала, а пока запомним только то, что он является одним из главных средств описания взаимодействующих последовательных процессов -

основной идеологии транспьютерных систем, к которым мы сейчас и переходим.

Транспьютер и остальные

Концепция ВПП и тесно связанный с ней язык Оккам оказали определяющее влияние на проектирование транспьютера. Практически он создавался для реализации модели ВПП в железе. Однако пора, наконец, сказать, что же это такое.

В настоящее время транспьютером называют микро-ЭВМ, реализованную в одном кристалле и имеющую кроме процессора собственную локальную память и несколько каналов для соединения с другими транспьютерами с целью обмена информацией. Каналы транспьютера устойчиво называют линками (от английского link - связывать). Пока что транспьютеры широко выпускаются только британской фирмой Inmos, которая имеет существенный отрыв во времени от конкурентов, хотя в последнее время появились сообщения о разработках некоторыми американскими фирмами своих транспьютерных микросхем.

Теперь кратко рассмотрим устройство и возможности типичного представителя транспьютеров фирмы Inmos - СБИС IMS T800.

Транспьютер T800 имеет внутри своего кристалла общую 32 - разрядную шину, на которую навешаны: главный 32- разрядный процессор (тактовая частота обычно - 20 МГц) ; сопроцессор для поддержки арифметики с плавающей точкой; 4 килобайта внутренней сверхбыстродействующей памяти (время доступа - один такт процессора = 50 нс); интерфейс внешней памяти, который позволяет подключать к транспьютеру до 4 ГБ внешней динамической памяти (адрес равен слову процессора, время выборки обычно составляет три такта процессора); общесистемный блок (сервисная аппаратура) и четыре линка (интерфейсы связи с другими транспьютерами), которые позволяют осуществлять последовательный обмен данными со скоростями 20, 10 и 5 Мбит/сек (на выбор; скорость зависит от дальности связи). Блок-схема устройства транспьютера T800 приведена на рисунке..

Чтобы состыковать между собой линки различных транспьютеров, достаточно соответствующие ножки микросхем соединить обыкновенным проводником (обычно через всевозможные перемычки и микросхемы-коммутаторы), и после чего транспьютеры будут успешно обмениваться информацией, даже если они разного типа (но фирмы Inmos) и разной тактовой частоты. Таким образом, из множества транспьютеров можно построить мощный многопроцессорный суперкомпьютер, представляющий из себя некую транспьютерную сеть с меняющейся топологией, которую с помощью перемычек (в старых конструкциях) или переконфигурацией специальных микросхем-коммутаторов (о которых ниже) можно легко приспособить под решаемую вычислительную задачу.

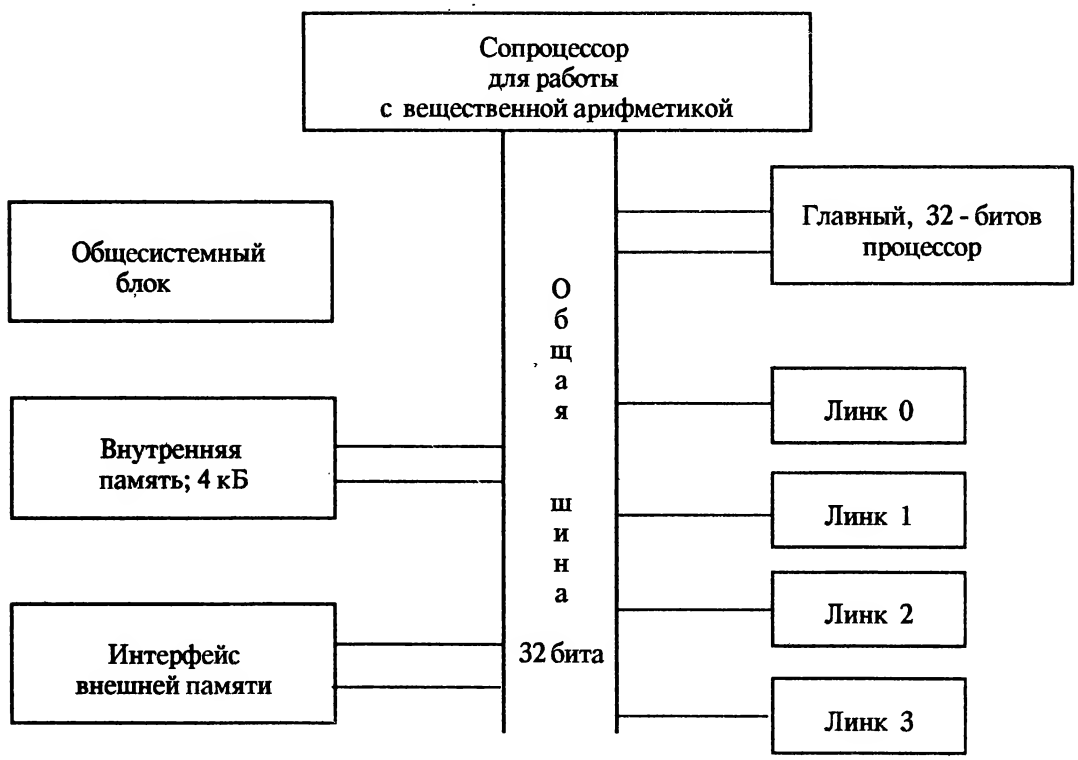
Важной особенностью модели T800 является возможность выполнения на одном транспьютере нескольких независимых процессов в режиме квантования времени. Это обеспечивается наличием в центральном процессоре аппаратного планировщика заданий, который позволяет создавать процессы двух приоритетов: высокого и низкого. Процессы, выполняемые на одном транспьютере, могут иметь между собой сколько угодно каналов, а процессы на разных транспьютерах - не более числа связывающих их линков.

О том, как работает система приоритетов, и о других деталях работы транспьютера IMS T800 можно прочесть в статье [2]. А мы добавим еще несколько слов о вычислительных способностях транспьютера.

Согласно данным фирмы Inmos, 20- мегагерцевый транспьютер IMS T800 на тесте " Ливерморский цикл " обеспечивает производительность 1,5 миллионов операций с плавающей точкой в секунду (1,5 Mflops). Для операций с целочисленной арифметикой IMS T800 имеет максимальную производительность около 10 милионов операций в секунду (10 Mips). В целом, если сравнивать IMS T800 с другими известными процессорами, то этот транспьютер по своей производительности слегка превосходит процессор Intel 386, работающий совместно с сопроцессором Intel 387. Отметим, что в настоящее время широко доступны транспьютеры с тактовой частотой 25 и 30 МГц и более старших моделей (T 805).

Кроме транспьютеров фирмой Inmos выпускаются также микросхемы-интерфейсы, позволяющие подключать транспьютерные сети к различным ЭВМ через их внутренние шины (например IBM PC). Обычно это делается

Рис.



подключением линка одного из транспьютеров к такой микросхеме и далее к шине ЭВМ. В этом случае подключенный к ЭВМ транспьютер называют корневым (root transputer) , и через него происходит весь обмен данными между транспьютерной сетью и ЭВМ (и ее периферией!).

Кроме того, стоит еще упомянуть микросхемы - коммутаторы, которые позволяют произвольным образом соединять между собой подключенные к ним линки транспьютеров. Перекоммутация осуществляется программным путем по командному линку микросхемы, что позволяет менять по необходимости конфигурацию транспьютерной сети прямо во время работы программы. Отметим, что данная возможность использовать изменение собственного суперкомпьютера во время его работы не исследована, в плане создания новых алгоритмов вычислений, даже теоретически.

Programmer

Рост потребности в вычислительной мощности приводит к тому, что приходится брать не только скоростью, но и числом процессоров. Поэтому в скором времени всем нам придется осваивать параллельные вычисления

В настоящее время всеми ведущими фирмами - производителями транспьютерного оборудования принята идеология материнских плат (motheboard), то есть выпускается плата, приспособленная под данный тип ЭВМ (с необходимым интерфейсом), на которой смонтировано какое-то базовое оборудование (в том числе возможно и микросхемы - коммутаторы) и имеются несколько специальных гнезд (слотов, посадочных мест), куда можно вставлять стандартные модули, содержащие уже собственно транспьютерную аппаратуру. Подобные модули называют "трамами" (от слов TRAnsputer Module). Они имеют стандартизированные размеры и позволяют монтировать на материнских платах вычислительные устройства самой различной конфигурации, поскольку набор выпускаемых трамов очень широк и может удовлетворить самые разнообразные запросы. Для примера перечислим обзорно трамы, выпускаемые одним из лидеров по производству транспьютерного оборудования - британской фирмой Transtech Parallel Systems. Это прежде всего (кроме самых различных материнских плат) модули, содержащие транспьютер с подключенной к нему внешней памятью (от 128 КБ до 16 МБ) с разным временем доступа. Имеются модули (TTM100 и TTM110), позволяющие подключать к транспьютерной сети очень известный процессор Intel 860. Кроме того, имеется широкий набор контроллеров, осуществляющих интерфейсы: SCSI (трам TTM11-2); RS232, RS432 и Parallel (Centronics) (трам TTM13) . Графические контроллеры TTG1 и TTG3 позволяют осуществлять управление самыми различными дисплеями и делать с ними все, что угодно. Модуль TTG-F (Framegraber) позволяет осуществлять ввод изображения с видеосигнала в транспьютерную сеть. Имеются также различные ЦАП, АЦП. Для всех устройств разработано необходимое ПО.

При помощи выпускаемого сейчас транспьютерного оборудования пользователь может, словно из деталей детского конструктора, собирать в своей ЭВМ (и не только в своей) самые различные вычислительные и управляющие комплексы практически произвольного назначения и очень высоких возможностей. И эти возможности далеко еще не исчерпаны, причем они постоянно расширяются [3].

На этом мы закончим краткий обзор транспьютерной аппаратуры и скажем несколько слов о транспьютерном программном обеспечении.

В настоящее время, кроме упоминавшегося Оккама (для него кстати существуют мощные интегрированные среды), разработаны компиляторы с языков Параллельные Си, Фортран, Паскаль, все эти языки не слишком сильно отличаются от уже известных аналогов, так что имея представление о ВПП и транспьютерах, переучиться не сложно. Существуют также компиляторы с языков STRAND88, Ада, Параллельный Пролог, Модула-2 и конечно же транспьютерный ассемблер. Для многих языков разработаны отладчики (TBUG , DEBUG и другие). Кроме того, имеются различные распределенные операционные системы для транспьютерных систем, самой известной из которых является ОС Helios. Подробнее о транспьютерном программном обеспечении можно прочитать в [4].

Транспьютеры в России

В нашей стране всегда проявлялся интерес к параллельным вычислениям и ЭВМ для них как к средству для достижения высокой и сверхвысокой производительности при невысокой стоимости оборудования. Однако отечественные разработки высокопроизводительных вычислительных комплексов сдерживаются низким уровнем элементной базы, да и значительными достижениями в идеологии таких ЭВМ мы похвастаться не можем. Этим и объясняется большой интерес как разработчиков, так и пользователей ЭВМ к многообещающей транспьютерной технологии, которая, кроме того, может стать каким-то стандартом в области параллельных вычислений и даже для персональных параллельных компьютеров, что очень важно для рынка ПО.

В России успешно работает общественная некоммерческая организация " Российская транспьютерная ассоциация " (РТА) , призванная содействовать развитию и распространению транспьютерной технологии. Она была создана в октябре 1990 года по инициативе многих вузов, научных и промышленных организаций, среди которых следует упомянуть: институт прикладной математики РАН им. М.В. Келдыша ; Всесоюзный институт проблем вычислительной техники и информатики ; НИИ " Квант " МРП (разработчик высокопроизводительной ВТ) ; СП " Инкомпекс " и другие. Председателем совета РТА является член-корр. АН СССР, профессор В.К. Левин. Ассоциация осуществляет широкое информационное обслуживание своих членов, проводятся конференции и семинары, издается "Вестник Российской Транспьютерной Ассоциации", установлены контакты с " Британской транспьютерной инициативой" (SERC/DTI Transputer Initiativ), - координатором развития транспьютерной технологии в Британии, и рядом британских университетов, занимающихся этой тематикой . Контактный телефон РТА - 153-90-68 .

Транспьютеры используются во многих организациях, особенно в научных и связанных с авиацией, однако назвать их конкретно по некоторым причинам затруднительно. Существует несколько транспьютерных центров (в основном в Москве), в которых ведется разработка транспьютерного программного обеспечения и продается на Запад.

Одной из главных проблем для российских транспьютерщиков является несомненно добывание аппаратуры. Однако поскольку ее не так уж много и надо (одна материнская плата, которая вставляется в обычную IBM PC, несколько трамов и ПО), то эта проблема вполне решаемая, тем более что отдача велика. Проконсультироваться по вопросам приобретения транспьютерной аппаратуры можно по телефону 499-15-00. Кроме того, в Москве есть Центр Параллельных Систем и Технологий, который проводит платные учебные курсы по всем аспектам транспьютерной технологии и оказывает другие услуги. Контактный телефон Центра - 928-21-38 .

В целом, если говорить о России, явно намечается постепенная ориентация, как и в странах Европы, на транспьютерную технологию как на обладающую, пожалуй, самым высоким потенциалом и универсальностью. На Западе в последнее время отмечается бурное развитие этого направления, и можно надеяться, что и мы не останемся в стороне. Вообще складывается впечатление, что следующим этапом после освоения персоналок будет соединение их с транспьютерами, и это будет видимо обычным явлением. Пока мы находимся в начале этого интересного пути, но перспективы огромны, и это вселяет уверенность. ■

Литература.

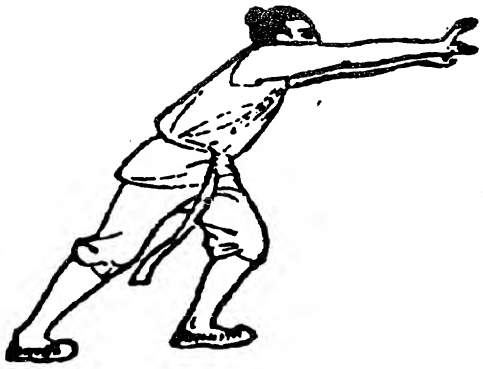
[1] Ч.Э.Р. Хоар . Взаимодействующие последовательные процессы ; М: Мир, 1989.

[2] В.А. Лопатин, К.К. Хачатурян . Транспьютеры - элементная база мощных параллельных компьютеров. Интеркомпьютер,1990, № 6.

[3] М.А. Маркин, В.А. Лопатин . Т900 Транспьютерная революция продолжается , Журнал Д-ра Добба, 1991, № 3.

[4] К.К. Хачатурян . Программное обеспечение транспьютерных систем. Интеркомпьютер, 1991, №№ 1-2.

ЗАЩИЩАЙТЕСЬ, СУДАРЬ!



И. Василенко
Е. Гулаков

Часть 1

Защита программного продукта от несанкционированного копирования - актуальная задача в связи с сохранением коммерческих и авторских прав фирм и разработчиков. По сведениям зарубежных специалистов экономический ущерб от "пиратского" копирования программного обеспечения составляет миллиарды долларов. Точные потери установить невозможно из-за отсутствия полных сведений о числе "пиратских" копий; считается, что с каждой программы их делается от 2 до 15.

С точки зрения профессионального программиста термин "защита от копирования" для IBM PC, работающей под управлением MS-DOS, достаточно условен, так как практически всегда возможно переписать информацию, находящуюся на дискете. Другое дело, что после этого программа может не выполняться. Таким образом, без санкции разработчика или фирмы-распространителя невозможно получить работоспособный программный продукт. То есть фактически "защита от копирования" - это создание средств, дающих возможность "защиты от несанкционированного выполнения".

Наиболее распространенной идеей, лежащей в основе большинства технологий защиты от копирования, является создание особо определяемых дискет. Их особенность заключается в том, что на дискете создается специально организованная метка - "маркант", которая используется как признак ее дистрибутивности. Функцию контроля "марканта" выполняет специальная часть защищаемой программы. После копирования средствами DOS защищаемого диска будет скопирована вся информаци за исключением "марканта". При выполнении программы ее контролирующая часть установит, что диск не дистрибутивный, и прервет выполнение программы. Тем самым программа как бы "привязывается" к своей дискете. Для создания "марканта" применяются программные и аппаратные средства, а также их комбинирование.

Другой способ предотвращения незаконного использования программ и данных заключается в хранении информации в кодированном, зашифрованном виде. В этом случае без знания ключа примененного преобразования работа с информацией невозможна.

Теперь давайте рассмотрим, что можно противопоставить "корсарам", рыщущим на волнах рынка software продуктов.

Перестановка в нумерации секторов

При подготовке новой дискеты к работе она форматируется, т. е. определяются количество дорожек, длина сектора, количество секторов на дорожке, нумерация секторов (то есть формируется ID маркер) и производятся другие операции. Если эти действия осуществляются с установкой параметров (длины сектора, нумерации секторов, величины межсекторного промежутка и др.), отличных от принятых по умолчанию для системы MS-DOS, то такой процесс будем называть нестандартным форматированием дискеты.

Один из методов зашиты от копирования основывается на перестановке номеров секторов на дорожке, то есть вместо обычной последовательности 1,2,3,4,5,6,7,8,9 вводится, например, 1,5,3,7,9,8,6,2,4.

При выполнении программы на скопированной дискете ее контролирующая часть определяет порядок следования секторов на заданной дорожке. Так как дискета, на которую осуществлено копирование, была отформатирована обычными средствами, то нумерация секторов установлена последовательная.

В дальнейшем контролирующая часть программы осуществляет сравнение вычисленного порядка следования секторов с установленным на дистрибутивной дискете. И так как порядок номеров не совпадает, то выполнение программы прекращается.

Введение одинаковых номеров секторов на дорожке

Другой схемой защиты, основанной на идее нестандартного форматирования, является способ, при котором часть секторов на определенной дорожке нумеруется одинаково. Например, 1,2,3,3,3,6,7,8,9. В эти секторы записываются некоторые различные данные.

Контролирующая часть защищаемой программы должна определить, имеются ли на дискете несколько секторов с одинаковым номером. Для этого она посылает запрос чтения данных из сектора, номер которого повторяется (в нашем примере из сектора 3). В этом случае контроллер НГМД при выводе головки чтения/записи на сектор 3 может считать любой сектор с данным номером. Этот процесс повторяется заданное количество раз подряд, и очередные считанные данные сравниваются с полученными ранее. В случае их различия делается вывод о наличии на дискете секторов с одинаковым номером. Если же за заданное количество повторений цикла N различия в данных не найдены, то делается вывод о единственности сектора с номером 3.

Так как дискета, на которую осуществлено копирование, была отформатирована обычными средствами, то нумерация секторов на всех дорожках устанавливается последовательной. Контролирующая часть защищаемой программы организует проверку на наличие нескольких секторов с одинаковыми номером. И так как каждый номер сектора на заданной дорожке присутствует только один раз, то выполнение программы прекращается.

Этот процесс представлен блок-схемой на рис.1.

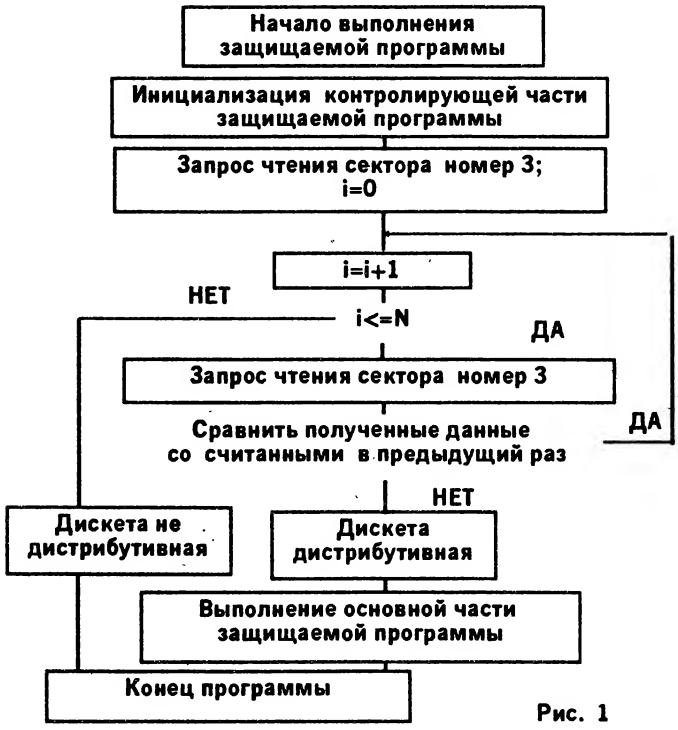


Рис. 1

Введение межсекторных связей

В одном из методов защиты от копирования используется модификация изложенного выше способа, применяющего организацию нескольких секторов с одинаковым номером на заданной дорожке.

Суть метода в следующем. На выделенной дорожке дистрибутивной дискеты путем применения специальной программы организуются несколько секторов с одинаковым номером и в них записываются некоторые различные данные. Пусть, например, номера на 39-й дорожке заданы как 2,2,1,4,5,6,7,8,9 и в первом секторе с номером 2 записано "С", а во втором секторе с номером 2 - "В". Берется еще одна дорожка, например 38, и сектор на ней, например 1-й, и в него записывается некоторая информация, пусть "А". В контролирующей части защищаемой программы организуется чтение сектора 1 на 38-й дорожке и сразу же запрос чтения сектора 2 на 39 -й дорожке. Тем самым всегда обеспечивается переход к чтению информации из второго сектора с номером 2 на дорожке 39, то есть мы прочитаем биграмму "АВ" с дистрибутивной дискеты.

Пусть было произведено "пиратское" копирование дискеты с защищаемой программой на обычно отформатированную дискету. Контролирующая часть программы считывает 1-й сектор с 38-й дорожки. Далее идет запрос чтения сектора 2 на 39-й дорожке. Так как секторы на "пиратской" дискете пронумерованы стандартно, то есть 1,2,3,4,5,6,7,8,9, то после запроса сектора 2 будет получено значение "С". Следовательно будет прочитана биграмма "АС".

Таким образом, контролирующая часть защищаемой программы устанавливает, что если дискета не дистрибутивная, то прочитывается биграмма "АС", означающая, что считан сектор 2 с 39-й дорожки, который находится на ином физическом расстоянии от начала дорожки по сравнению с расстоянием для сектора 2 на дистрибутивной дискете, когда считывается биграмма "АВ". Программа прерывается.

Изменение длины секторов

Еще одна схема защиты, основанная на методе нестандартного форматирования, использует изменение длины сектора. Напомним, что стандартная длина сектора, с которой работает MS-DOS по умолчанию, - 512 байтов, при этом на дорожке размещаются 9 секторов. В процессе специального форматирования дискеты на заданной дорожке длина секторов устанавливается либо 128, либо 256 байтов.

Введенное изменение длины секторов на заданной дорожке определяется контролирующей частью защищаемой программы. Это осуществляется, например, путем обращения к сектору с максимальным номером, либо проверкой того, что длина секторов на указанной дорожке 128 (или 256) байтов.

Стандартно отформатированная дискета, на которую проведено "пиратское" копирование защищаемой программы, имеет длину секторов на всех дорожках 512 байтов (по 9 секторов на дорожке). В ходе выполнения защищаемой программы ее контролирующая часть осуществляет провер-

ку длины секторов на заданной дорожке. Если контролирующая часть использует способ обращения к сектору с максимальным номером (большим 9), то такой сектор найден не будет (так как их только 9) и выполнение программы прекращается.

Аналогично работает контролирующая часть и в способе, в котором осуществляется проверка, что длина секторов 128 (или 256) байтов.

Изменение межсекторных промежутков

Таблица базы дисков, с помощью которой частично управляется работа НГМД, содержит параметр, определяющий величину межсекторного промежутка. Выбирается дорожка на дискете, которая специальной программой форматируется заново с измененным значением межсекторного промежутка в таблице базы дисков, что ведет к нестандартному расположению секторов.

Контролирующая часть защищаемой программы при обращении к выбранной дорожке производит ее анализ и делает вывод, является ли дискета дистрибутивной.

Использование дополнительной дорожки

При этом способе защиты используется дополнительная, 41-я дорожка на дистрибутивной дискете. Заметим, что на гибком диске имеется место для размещения еще трех дорожек, сверх принятых 40. Дисковод позволяет вести на них запись данных. Поэтому на дополнительной 41-й дорожке на дистрибутивной дискете хранится информация для контролирующей части защищаемой программы.

При копировании на стандартно отформатированную дискету дополнительно введенная 41-я дорожка воспринята не будет, что и обнаруживается контролирующей частью защищаемой программы.

Введение логических дефектов в заданный сектор

Сначала необходимо сказать несколько слов о физическом и логическом строении дорожки на гибком диске. Как известно, начало дорожек на дискете отмечено индексным отверстием. Каждый сектор на дорожке имеет две части: секторный ID маркер и собственно данные. ID маркер имеет шесть байтов и дополнительные байты, которые идентифицируют его для контроллера накопителя на гибких магнитных дисках (НГМД) как секторный маркер, а не как данные. Эти шесть байтов следующие.

1-й. Номер дорожки с 0 -го по 39 -й включительно.

2-й. Номер стороны или номер головки. Верхняя сторона обозначается 0, нижняя - 1.

3-й. Номер сектора на треке с 1 -го по 9 -й включительно.

4-й. Размер сектора в байтах: 0,1,2 или 3 обозначают соответственно 128,256,512 или 1024 байта в секторе. По умолчанию формат DOS имеет 512 байтов в секторе.

5-й и 6-й. CRC байты для контроля при помощи циклического избыточного кода возможных искажений.

Четыре первых байта секторного ID маркера обозначаются как CHRN. CRC байты фактически содержат 16-битовую контрольную сумму CHRN байтов, которая вычисляется, когда сектор записывается. Когда же сектор читается, контрольная сумма перевычисляется для прочитанных из CHRN данных и сравнивается с данными из байтов CRC. Любое расхождение вызывает CRC ошибку ID маркера.

Следующими за ID маркером идут несколько меточных байтов маркера начала данных (BOD) и 512 байтов собственно данных. Еще два проверочных байта CRC следуют за данными. Они содержат контрольную сумму 512 байтов данных, вычисленных при записи сектора. Расхождение между контрольными суммами, записанными в CRC байтах и вычисленными при чтении, определяется как CRC ошибка данных. После CRC байтов первого сектора имеется межсекторный промежуток, затем идут другие секторы и межсекторные промежутки.

Теперь вам будет понятна идея метода защиты, основанного на использовании CRC- ошибки данных. Ее возникновение достигается следующим образом. Стандартная операция записи данных в сектор включает шесть шагов.

1. Включение мотора НГМД и короткое ожидание его разгона.
2. Выполнение операции поиска заданного сектора и ожидание прерывания, указывающего на завершение данной операции.
3. Инициализацию микросхемы прямого доступа в память (DMA) 8237 для пересылки данных из памяти.
4. Посылку команды записи микросхеме контроллера НГМД 765 и ожидание прерывания, указывающего на окончание пересылки данных.
5. Получение информации о статусе контроллера НГМД.
6. Выключение мотора.

На дистрибутивной дискете при выполнении операции записи в определенный сектор во время четвертго шага изложенной выше процедуры производится кратковременное отключение мотора НГМД, что приводит к искажению записываемых данных. Контрольная же сумма данных в CRC байтах будет иметь значение, которое было получено по исходным передаваемым данным. Так что после чтения данных из этого сектора и подсчета их контрольной суммы полученное значение будет отличаться от содержащегося в CRC байтах. Таким образом, контролирующая часть защищаемой программы при попытке чтения заданного сектора обнаружит CRC ошибку данных и определит дискету как дистрибутивную.

При копировании ранее искаженные данные на заданном секторе воспринимаются как истинные и по ним подсчитывается соответствующая контрольная сумма. Следовательно при чтении данного сектора на скопированной дискете CRC ошибка данных не возникает, и контролирующая часть защищаемой программы дискету отвергает.

Введение ограничений на использование программного обеспечения

В одном из способов защиты программного обеспечения от незаконного использования применяются ограничения по

времени его эксплуатации, количеству запусков, его перемещению для использования на других машинах.

Суть введения ограничения на время эксплуатации программы заключается в том, что при очередном запуске контролирующая часть программы производит сравнение показаний таймера системы с заданной разработчиком датой. Если указанное время истекло, то производится самоуничтожение данного программного продукта либо иные действия, предусмотренные для такой ситуации.

Еще одним способом ограничить использование программного обеспечения является введение счетчика запусков. Такой счетчик часто располагают либо в CMOS памяти машины (для IBM PC/AT), либо в загрузочном секторе (вместо информации о версии DOS); либо в FAT (в первом элементе таблицы, обычно заполненном кодом FFF), либо во внешнем файле. Уменьшая значение счетчика при каждом запуске, контролирующая часть программы следит за ним и при достижении нулевого значения выполняет предусмотренные разработчиком действия.

Персональные компьютеры не имеют уникально заданного серийного номера, поэтому привязка программного обеспечения возможна только к классу машин. Для этого часто применяется способ, в котором после запуска программы ее контролирующая часть определяет дату изготовления BIOS, установленной на данной IBM PC. Таким образом, число машин, на которые можно переместить программное обеспечение "пиратским" путем, резко снижается.

Изменение параметров дисковода

Одной из схем создания защиты от копирования используется другая скорость вращения диска. Стандартня скорость вращения - 300 об/мин. Если ее понизить, скажем, до 280 об/мин, в то время как данные для записи передаются с прежней скоростью, то это понижение увеличивает плотность хранения данных на дискете, и на каждой дорожке образуется место для дополнительного десятого сектора. Это обстоятельство является основой защиты, так как работа с информацией, записанной на дискете, вращающейся с новой скоростью, возможна только при соответствующей модификации параметров дисковода.

Технология "ослабленных" битов

Способом записи информации на дискету является ее представление в виде 0 или 1. Данный метод защиты использует запись некоторого участка дистрибутивной дискеты с неопределенным уровнем сигнала. Таким образом, получается участок "ослабленных" битов. Данные с этого участка при их чтении несколько раз подряд будут восприниматься различным образом ввиду того, что сигнал может преобразовываться только в два дискретных значения - 0 или 1.

При копировании дистрибутивной дискеты участок "неопределенных" данных примет какое-то фиксированное значение.

Таким образом, контролирующая часть защищаемой программы организовывает чтение указанного участка несколько раз подряд. Если получаются одинаковые данные при всех запросах чтения, следовательно дискета не дистрибутивная и программа прерывается.

Физическая маркировка дискеты

Одной из наиболее известных технологий защиты от копирования является физическая пометка дискеты лазерным лучем путем ее прожигания. Такой дефект приводит к искажению данных в секторе и возникновению CRC ошибки данных при его чтении. При копировании пробивка не воспроизводится на принимающей дискете, хотя данные переписываются. Таким образом, CRC ошибка данных при чтении сектора с копии не возникает.

Контролирующая часть защищаемой программы, производя чтение соответствующего сектора, это устанавливает и производит действия, предусмотренные разработчиком (прерывание выполнения программы, ее уничтожение и т.п.).

Проверка присутствия пробивки может производиться контролирующей частью программы и другим, более надежным способом. Она производит попытку записи в место, где должно находиться отверстие. Неудача - явный признак того, что дискета дистрибутивная. Если же запись производится успешно, следовательно поверхность в данном месте не повреждена и дискета не дистрибутивная.

Рассмотренная технология применяется фирмой Vault Corporation в средстве PROLOK и его дальнейших модификациях.

Применение физического защитного устройства

Многие аппаратно - программные способы защиты основаны на том, что в компьютер добавляется специальное физическое защитное устройство. (Оно подключается, например, к порту ввода-вывода). При запуске защищаемой программы ее контролирующая часть обращается к этому дополнительному устройству, проверяя его присутствие. Если оно не найдено (некоторые устройства еще и формируют код ответа, который затем анализируется), то производится останов программы либо какие-то иные действия (например, уничтожение информации на дискете или

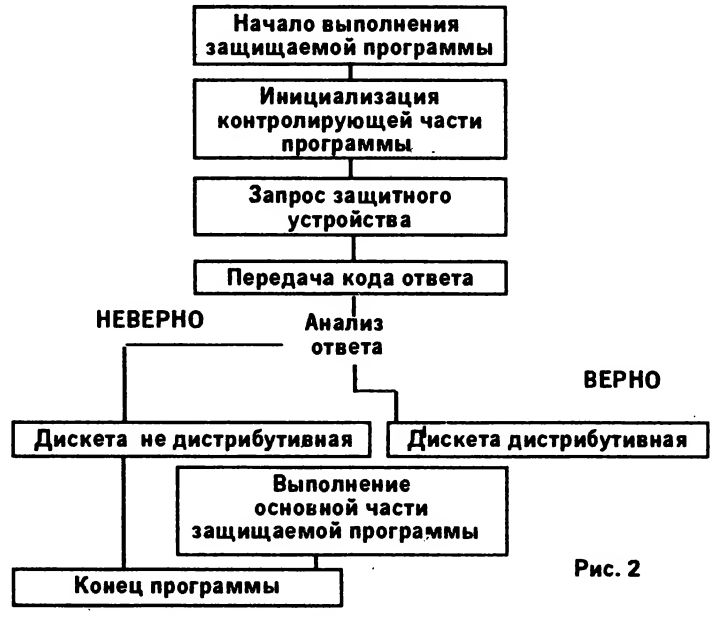


Рис. 2

"винчестере"). Иллюстрация работы такого способа защиты представлена блок-схемой на рис.2.

Общий принцип работы компьютера в этом случае следующий. После запроса на выполнение защищаемой программы происходят ее загрузка в оперативную память и инициализация контролирующей части. На физическое устройство защиты, подсоединенное к компьютеру через порт ввода-вывода (либо другим способом), посылается запрос. В ответ формируется код, посылаемый через микропроцессор в оперативную память для распознавания контролирующей частью программы. В зависимости от правильности кода ответа программа либо прерывается, либо выполняется.

Кодирование информации

Целая группа средств, защищающих от незаконного использования информации, применяет криптографическое закрытие информации. Эти средства реализуются как программно, так и технически. Криптографическая защита программ (данных) заключается в том, что с помощью специального алгоритма шифрования информация преобразуется в такой вид, который предотвращает ее распознавание и использование.

Наиболее широкое коммерческое применение нашли два алгоритма шифрования - DES и RSA. Первый разработан по инициативе Национального бюро стандартов США фирмой IBM в начале 70 -х годов и принят в качестве стандарта шифрования данных. При корректной реализации и надлежащем использовании DES алгоритм обеспечивает высокую степень криптографической защиты компьютерных данных.

Второй наиболее распространенный алгоритм - RSA - основан на так называемой схеме с открытым ключом. Суть ее заключается в следующем. Шифрование информации производится с использованием открытого несекретного ключа. Для преобразования зашифрованных данных в исходные применяется секретный ключ пользователя. Стойкость данного алгоритма основана на том, что без знания секретного ключа получение по зашифрованной информации открытой является весьма трудоемкой вычислительной задачей, которая не решается за приемлемое время.

Опрос справочников

Еще один способ защиты от несанкционированного использования информации называется опросом справочников. Он не обеспечивает стойкой, долговременной защиты программного обеспечения, но может быть легко реализован и очень удобен в работе. При его применении пользователь не обязан работать с дистрибутивной дискетой, можно и с ее копиями. Работа программ, защищенных на основе этого метода и их копий, делится на два этапа. Первый: получение от пользователя информации из некоторого источника, называемого справочником, и сравнение ее с эталонной. (То есть программа проверяет, действительно ли введенная информация находится в справочнике.) Второй: работа самой программы или ее копии. При этом необходимо сделать все возможное, чтобы текст справочника был доступен только законному пользователю, так как доступ к справочнику в этом методе снимает все преграды для работы программы. Описанная возможность работы с копиями, недоступная при реализации других методов защиты данных, заставляет изготовителей программного обеспечения применять этот способ в своих разработках.

В простейшем случае справочником является обычный текстовый файл, который пользователь располагает либо на дискете, либо на жестком диске, либо имеет его в качестве распечатки. Его можно сделать скрытым, если он расположен на магнитном носителе. Вместо текстового файла можно использовать служебную информацию, которая характеризует определенный класс машин. Однако в этом случае придется преодолевать дополнительные сложности при доступе к такой информации.

Сравнение введенной информации (пароля) с соответствующей информацией из текста справочника может быть осуществлено двумя способами.

1. Путем поиска информации в текстовом файле, расположенном на магнитном носителе, по указанным координатам и сравнения ее с введенной. Например, программа выдает запрос: "Введите пятое слово третьего абзаца седьмой страницы." Пользователь вводит слово. Программа считывает из текста справочника пятое слово третьего абзаца седьмой страницы и сравнивает его с введенным.

2. Путем сравнения слова из определенной части текста, представленного распечаткой, с данными, содержащимися в защищающей программе. Этот случай является как бы обратным к случаю 1. Программа содержит информацию о том, что в тексте справочника такие-то слова о том находятся в таких-то местах (например, слово "программа" является пятым словом третьего абзаца седьмой страницы, слово "справочник" - первое слово второго абзаца пятой страницы и т.д). В ответ на запрос программы, описанный в первом способе, пользователь в тексте справочника по указанным координатам ищет слово и вводит его. Программа сравнивает полученное слово с информацией, которой она располагает.

При небольшом по объему справочнике, а также малом запасе опрашиваемых парольных слов целесообразно список всех парольных слов разместить совершенно открыто в качестве файла на специальной дискете. Это позволит пользователю не держать в памяти информацию о том, какое слово где находится, и не тратить время на поиски нужного слова в справочнике, а снимать защиту самым элементарным образом, опробуя каждое слово из созданного списка. Но если эта дискета попадет в руки к постороннему лицу, то оно будет благодарно вам за возможность просто ознакомиться с вашими секретами.

Однако просмотр перед каждым запуском программы списка паролей и опробование каждого из них занимают довольно много времени. К тому же в таких системах часто возникают ошибки из-за неоднозначности решения: выделять или нет заголовок как отдельный абзац. Например, взяв пятое слово из третьего абзаца в качестве пароля, вы не можете запустить программу, так как автор выделил заголовок в отдельный абзац и требуемое слово находится не в третьем, а во втором абзаце текста, если рассматривать его без заголовка.

Из-за указанных недостатков описанный выше механизм чаще всего используется в качестве составной части многопрофильных систем защиты от копирования. ■

Фирма ИНСАЙТ **INSIGHT Comp**

Официальный дистрибьютер

CLARION SOFTWARE

Впервые! Фирма ИНСАЙТ начала поставку за рубли лучшей в мире интегрированной среды разработки программного обеспечения.

К фирменному комплекту поставки прилагается документация на русском языке. Проводятся учебные курсы, действует CLARION User Group, работает BBS, для зарегистрированных пользователей действует "горячая линия".

Предлагаем также Pascal, Modula-2, C++ **TopSpeed-***компиляторы фирмы* **JPI**
Магнитные носители фирмы **3M** * *Принтеры* **STAR** *Micronics*
Программное обеспечение фирм **Borland** Int. & **Symantec** Corp.

103460, Москва, Панфиловский просп., корп. 1205, фирма "ИНСАЙТ"
тел. (095) 532-29-43, 531-82-30 факс 532-29-43 E-Mail: puh@pczz.msk.su | 2:5020/11

ТРОЛЛЬ
Официальный представитель Horstmann Software Design Corporation

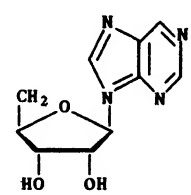
horstmann software design

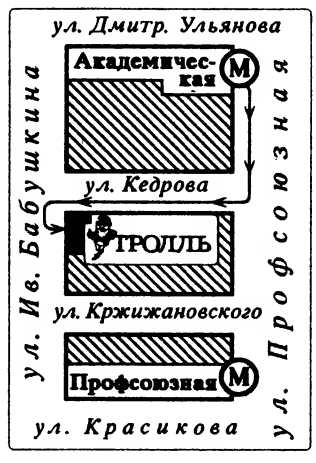
Редактор текста для научных целей

ChiWriter 4

Новая версия — новые возможности!

$$\sqrt{E\left(\frac{du}{dt}\right)^2 + 2F\frac{du}{dt}\frac{dv}{dt}}$$





Наш демо-центр: Москва, ул. Ив. Бабушкина, д. 11/2,
ИНОТЕХ-Прогресс: МП «Тролль». Тел.: (095) 124-9486

Лемминги идут на север

А. Лукашук

Фирма-разработчик : (C) Psygnosis Ltd.
Компьютер : IBM PC XT/AT; PS2; AMSTRAD PC1640/PC1512
Графика : CGA; Tandy; EGA; VGA
Звук : Adlib Sound; IBM Music Feature; Speaker
Управление : Keyboard; Joystick; Mouse

Удивительный мир животных всегда служил неисчерпаемой кладезью идей и сценариев сказок, фильмов и различных художественных постановок. Неплохо преуспели здесь и компьютерные аниматоры. На этот раз предметом их вожделений стал очень интересный и практически неведомый у нас зверек. Именем этой божьей твари была названа эта игра - Л Е М М И Н Г И. Удивительно то, насколько сильно авторы игры абстрагировались от животного как такового, отметив лишь в названии главного виновника этой широко распространенной игры.

Лемминги - это маленькие неприхотливые существа, напоминающие мышек или хомячков. Основная отличительная черта их - поразительное чувство коллективизма. Самопожертвование ради общего блага,-такие понятные и ностальгически потерянные нами инстинкты (а все проклятый капитализм),проход в одиночку сложного и опасного пути с одной лишь целью - провести всех "страждущих и неимущих" (нежданный каламбур) к заветной цели.

Общая идея игры поражает своей незамысловатостью: привести всех (или заданное количество) леммингов "домой", используя разрешенные средства. Более подробно о средствах будет сказано ниже. Лемминги, выпадая из своего убежища, начинают брести направо(именно направо - это принципиально).Необходимо привести их "домой". На игровом экране Вы без труда определите конечный пункт похода("дом") - это будут красивые, открытые для всех ворота. Будьте внимательны, в некоторых уровнях Вам не удастся сразу найти "домик", но не отчаивайтесь, он есть на всех без исключения уровнях и бывает замурован в скалу. Надо просто прогрызть ее, и Вы получите доступ к искомой цели. Препятствиями на Вашем пути может стать ландшафт местности и ловушки, любезно предусмотренный разработчиками (типа огнедышащих драконов, падающих колонн, кусающихся растений и прочих развлечений). Не стоит отчаиваться: предоставленные Вам средства всегда обеспечат Вам успех в задуманном мероприятии.

Вы можете управлять лишь курсором, направляя его на выбранного Вами лемминга, но чтобы лемминг обрел необходимую Вам функциональность, надо выбрать одну из разрешенных функций в нижней части экрана. Выбор можно производить как соответствующей функциональной клавишей, так и курсором. Необходимо отметить, что количество назначений той или иной функции показано на изображении функциональной клавиши.

Будьте внимательны: для прохождения уровня выделяется строго определенное время.

С самого начала ...

Основное меню игры представлено следующими клавишами :

F1	- запуск игры
F2	- выбор нового уровня (необходимо ввести код уровня)
F3	- вкл/выкл звука
F4	- выбор устройства управления(клавиатуры, "мыши", джойстика)
ESC	- выход в DOS
Uparrow	- увеличение уровня сложности
Down arrow	- уменьшение уровня сложности

В случае выбора клавиатуры как устройства управления, в Вашем распоряжении имеются следующие клавиши :

Q	- вверх
A	- вниз
O	- влево
P	- вправо
Пробел	- выбор

Если Вы выбрали "мышь", то Вам гораздо проще: движения "мыши" полностью отрабатываются курсором и левая клавиша мыши - выбор.

Джойстик управляет аналогичным образом и кнопка "Огонь" - выбор.

Перед выходом на новый уровень на экран выдаются количество леммингов, время прохождения уровня и необходимый процент сохранения для выхода на следующий уровень.

Клавиши управления игрой :

F1	- уменьшить скорость выхода леммингов
F2	- увеличить скорость выхода леммингов
F3	- выбор альпиниста
F4	- выбор парашютиста
F5	- выбор взрывателя
F6	- выбор стопера (блокера)
F7	- выбор строителя
F8	- выбор проходчика по горизонтали
F9	- выбор шахтера по диагонали
F10	- выбор диггера (копателя ямы) по вертикали
F11	- пауза
F12	- армагеддон - общее уничтожение леммингов (два нажатия)

ESC	- выход в основное меню
1	- выключение звука
2	- включение звука

Более подробно ...

Какие же возможности предоставляются Вам и Вашим младшим "братьям" ?

Сейчас наконец станет ясно, причем здесь лемминги... Сейчас, сейчас... Начнем по порядку. Очень полезной функцией, предоставляемой разработчиками лемингу, является возможность вскарабкиваться по отвесным стенам (клавиша F3), но сразу хочу предупредить, что вскарабкавшись на пригорок (бугорок), надо как-то оттуда слезать и "как ни странно" это предусмотрено разработчиками, - нажимай F4 и лемминг как заправский парашютист плавно спланирует на твердь земную и продолжит свой долгий и тяжелый путь.

Будем последовательны. Функциональная клавиша - F5... В игре, как в жизни, всегда бывают ситуации, когда любой ценой приходится добиваться намеченной цели, и часто ценой является жизнь лемминга. Но не стоит недооценивать недолгую, но порой очень необходимую жизнь Ваших героев. В большинстве ситуаций нехватка одного или нескольких леммингов приводит к тому, что уровень, "вымученный" Вами, не пройден. Итак, F5, самая жестокая функция леммингов. По F5 через 5 секунд происходит чудовищной силы взрыв, и в Вашем арсенале становится на одного бойца меньше.Рекомендуется использовать взрывы целенаправленно, так как не надо быть сильным пиротехником, чтобы взорвать перекрытие или проделать брешь в стене.

Следующая функциональная клавиша - F6 будет разумным дополнением к клавише F5. По F6 Ваш лемминг останавливается как вкопанный и загораживает проход остальным. Использовать эту функцию можно совместно с F5, например останавливаете лемминга около стены и взрываете его, получая тем самым проход в нужном месте. "Блокера" можно также использовать для перекрытия прохода основной массе леммингов с целью выделения одного, который, пройдя весь путь, построит проход остальным, а искомого блокера придется взорвать, чтобы освободить проход. Маленькая хитрость: блокера можно устранить, не взрывая - подкопав его, например, когда он находится на лестнице (эти функции описаны ниже).

В процессе игры Вам понадобится не только взбираться по стенам, но и переходить многочисленные провалы, пропасти и разнообразные ловушки. Для этого выделена клавиша F7. Лемминг начинает строить лестницу, но надо быть внимательным, так как строительный запал быстро пропадает, что сопровождается характерным звуком при укладке последних трех ступеней, и по необходимости можно продлить эту созидательную деятельность, назначив его снова строителем по F7, в противном случае лемминг спрыгнет с лестницы и продолжит свой путь к достижению намеченной цели. Старайтесь постоянно следить за количеством оставшихся лестниц, потому что F7 - наиболее часто используемая функция в игре.

Таблица 1. Коды уровней для LEMMINGS

LEMMINGS

FUN	TRICKY	TAXING	MAYHEM
1. IJJLDLCCCL	1. HCENNONPDW	1. MFMCEJLNFL	1. NJMFLGALHM
2. OHNLHCADCN	2. CIOLMFLQDU	2. FMCIKLMOFR	2. HMFLGMNMHX
3. JLLHCIOECW	3. CAJJLDMBEV	3. ICANLONPFT	3. MFLGAJLNHM
4. LDLCAJNFCK	4. MJHLDMCCEQ	4. CINNMFMQFY	4. FLGIJNMOHX
5. DLCIJNLGCT	5. NJNLICEDEV	5. GEKJLLHBGT	5. HGANNONPHK
6. HCANNNLHCW	6. JLDMBINEET	6. MKJNLHGCGO	6. GMOLONHQHW
7. CINNLDLICJ	7. LDMCAJNFEN	7. OJNLLFADGK	7. GAJJLDMBIN
8. CEJHMDLJCN	8. LICIJNLGEK	8. OHLDHGADGL	8. IJHLDMGCIU
9. MJHMDLCKCW	9. MCANLLDHER	9. HLDLGMNEGL	9. NHLDMGEDIV
10. NHMLHCALCT	10. CMNLLDMIEO	10. LDLGAJNFGS	10. HLDMGIOEIL
11. HOLHCINMCO	11. CEJJOLIJEY	11. DLGIJNLGGL	11. LLIGAJNFIJ
12. OLHCEKNNCO	12. IKJOLICKEO	12. LGANNLDHGY	12. DMGIJNLGIO
13. DLCIJNMOCM	13. OHMDMCALET	13. GINNLDLIGR	13. MGANNLDHIL
14. LCANLMDPCX	14. JODICINMEL	14. GAJHMLHJGV	14. GINNLDMIIU
15. CINLMDLQCQ	15. MDMCAJLNEU	15. IKHMDLGKGL	15. GAKHMDMJIV
16. CAKHNNHBDK	16. LMBIJNOOEY	16. NHMDLGELGL	16. MJJOLIGKIJ
17. IKJLNLBCDW	17. ICANNMLPEQ	17. HMDLGIOMGR	17. NJMDMGALIM
18. NJLNHCEDDT	18. CINLMDMQET	18. MDLGAKLNGK	18. HMDMGIOMIU
19. HLFLCINEDS	19. CAKHNNIBFN	19. LHGMJNOOGO	19. MDMGAJNNIO
20. LNHCAJNFDR	20. IJJLFMCCFR	20. LGANLMDPGP	20. LIGIKNOOIO
21. NHCIKLLGDJ	21. NHLFMCEDFQ	21. GINNOLHQGQ	21. MGANNMDPIU
22. LCENLLFHDV	22. JNNICINEFN	22. GEJHLFLBHP	22. GINNMOMQIN
23. CIONLFHIDJ	23. LFMCAJLFFO	23. IJJLFLGCHW	23. GEJHLFMBJS
24. CAKJMFHJDL	24. NMBIJNNGFS	24. OHLFLGADHS	24. MJHLFMGCJL
25. IJJMFHCKDT	25. MCENLLFHFY	25. HLFLGIOEHL	25. NJLFMGADJW
26. NJMFLCALDV	26. BINNNFIIFM	26. LFLGAJNFHV	
27. JONHCINMDT	27. CAJJMFMJFR	27. FLGMJLLGH Q	
28. MFHCAJNNDS	28. IJJONICKFQ	28. LGAOLLFHHK	
29. FLCMJLMODR	29. OHMFMCALFW	29. GIOLNNHIHJ	
	30. HMFMCIOMFP	30. GAKJMFHJHT	
		31. MJHMFLGKHR	

Таблица 2. Коды уровней для LEMMINGS2

LEMMINGS2

TAME	CRAZY	WILD	WICKED	HAVOC
1. BNMKOIUHJH	1. FLCAHVFBD	1. CEIPWDIJCF	1. UFMCAHTNDN	1. GEHPTFLBFF
2. IHRTDLCCAR	2. LCIHTTGBK	2. MHRWLICKCH	2. NICIHTWODM	2. MHPTFLGCFO
3. LRTDLCADAO	3. CALVTFHBJ	3. LPUDMCELCM	3. MCALVUFPDF	3. LPTFLGADFH
4. RTDLCILEAM	4. ILTTFLIBQ	4. PUDMCIMMCS	4. CILVUFMQD	4. RTFLGILEFS
5. TDLCAHVFAQ	5. AHRUFLIBE	5. UDMCAHVNCM	5. GAIPTDHBEL	5. TFLGAHVFFL
6. DLCIHVTGAT	6. HRUFLCKBN	6. LICMHVWOCP	6. IHRTDLGCEJ	6. FLGIHVTGFE
7. LCALVTDHAG	7. RUFLCALBK	7. MCALVUDPCS	7. LPTLLFADEL	7. LGALVTFHFR
8. CILVVLHIAF	8. UFLCILMBD	8. CILVUDMQCL	8. RVLHGILEEF	8. GMMVVNHIFF
9. CAHRUDLJAR	9. NHCEHVNBG	9. CAHRTFMBDO	9. VLHGAHVFEO	9. GAHRWNHJFS
10. IIPUDLCKAJ	10. LCIHVUOBF	10. IHRTFMCCDH	10. DLGIHTTGEP	10. IHRUFLGKFF
11. LPUDLCALAF	11. CAMTUFPBP	11. MRTNICADDJ	11. LGALTTDHEM	11. MRWFHFALFQ
12. RUDLCILMAQ	12. MMTWNHQBD	12. PTFMCIMEDM	12. GILTTDLIEF	12. RUFLGILMFL
13. UDLCAHVNAJ	13. AIPTDMBCK	13. VNICAIVFDN	13. GEHPUDLJEL	13. UFLGAHVNFE
14. LHCIHTWOAG	14. HRTDMCCCE	14. NICMIVVGDK	14. MHPUDLGKEE	14. FLGMHTUOFP
15. LCAMTUDPAO	15. RVLICADCH	15. ICALVVNHDS	15. MPUDLGALEO	15. LGALVUFPFK
16. CILVUDLQAI	16. VLICIMECR	16. CIMVTNIIDK	16. PWLHGILMEM	16. GILTUNHQFF
17. CAHRTFLBBL	17. DMCAHVFCD	17. CAHRUFMJDH	17. ULHGEITNEI	17. GAHPTDMBGR
18. IIPTFLCCBD	18. MCIITTGCL	18. IHRUFMCKDQ	18. DLGIHTUOEI	18. MHPTDMGCGO
19. LRTFLCADBR	19. CALVVLHCP	19. MPWNICALDS	19. HGELVWLPER	19. MPTDMGADGI

Следующие функции можно объединить по одному признаку: продвижению во внутрь породы. Эти функции выполняются нажатием клавиш F8, F9 и F10.

F8 - "проходчик" по горизонтали.

F9 - "шахтер" по диагонали.

F10 - диггер (копатель ямы) по вертикали.

Существует очень эффективный способ остановить друзей-копателей - назначить строителями. При этом меняются функции работы участников процесса: лемминг, усердно копая по вертикали, может быть назначен проходчиком или шахтером.

Следующая функция - F11. При кажущейся бесполезности пауза очень часто применяется для контроля ситуации, для управления игрой в разных экранах (т.е. когда игровое поле не умещается на одном экране дисплея).

Последняя и самая интересная клавиша - F12. Нужно быть великим разработчиком, чтобы предусмотреть такую клавишу, да еще так великолепно назвать ее : АРМАГЕДДОН. Я считаю, что нет необходимости рассказывать, что такое армагеддон (наверняка всем известно, что это идет из библейских приданий и обозначает всеобщий катаклизм,- конец света), нодействие этой клавиши совершенно потрясающее - общее уничтожение всех лемингов. Эта операция не так воинственна, как кажется, и применяется как правило для ускорения хода игры, например, Вы успешно прошли весь уровень и оставили на поле множество блокеров, очень долго назначать на каждого блокера по взрывателю (а часто взрывателей просто не хватает) или ждать окончания выделенного времени, проще и удобнее нажать два раза F12 (кстати именно два раза, что еще раз говорит о высокой продуманности всей игры, так как очень часто, желая быстро нажать на паузу, попадаешь на F12, и к нашей радости ничего не происходит).

Мы пропустили две клавиши - F1, F2. Лемминги в начале игры выпадают из дверей своего "дома" с определенной скоростью, клавишами F1 и F2 мы можем соответственно уменьшать и увеличивать частоту выхода лемингов. Число на изображении клавиши F1 обозначает минимальную частоту, на клавише F2 - максимальную.

По ключу <ESC> Вы можете выйти в основное меню игры.

Цифры 1 и 2 включают и выключают звук в процессе игры.

После прохождения уровня на экран выдается диагностическое сообщение о результатах прохождения, в случае успеха выдается код пройденного уровня. Знание кодов крайне необходимо, так как немного поработав, неизменно захочется поиграть, и так каждый день, но чтобы не проходить по несколько раз пройденные уровни, стоит лишь набрать код уровня и успешно продолжать игру с неутомимым энтузиазмом первопроходца. В принципе мы уже решили за Вас эту проблему. Ниже приведены коды уровней для всех рейтингов игры.

В табл. 1 даны коды уровней для игры LEMMINGS. А так как игра постоянно находится в динамике, мы описываем первые две серии лемингов, появившихся в нашем компьютерном пространстве. В следующей "серии" лемингов (сделанной, впрочем, по тому же сценарию, что не уменьшило ее привлекательности) "советские" (наши) хэкеры не сумели обойти фирменной защиты, и появился пароль (password) : BNMKOIUHJH. Затем все продолжается, как и раньше (табл. 2).

Ну что ж, друзья, Happy LEMMINGS !!! ■

24 июня в Московском коммерческом клубе состоялась встреча представителей фирмы Intel с журналистами компьютерных изданий.

Наибольший интерес вызвал рассказ Вице-президента фирмы Intel г-на Буше о результатах судебного процесса, который фирма Intel вела с фирмой Advanced Micro Devices (AMD). Предметом спора являются права на использование микрокода математического процессора Intel 80287 в аналогичном микропроцессоре фирмы AMD. В соответствии с соглашением 1976 года фирм Intel и AMD, последней разрешается копирование и использование микрокода лишь в макетных системах. На основании этого соглашения фирма AMD использовала микрокод и для своего математического сопроцессора, что по решению суда явилось нарушением прав на интеллектуальную собственность. Данное разбирательство является важным прецедентом, дающим гарантии фирме Intel в том, что ее усилиями не смогут воспользоваться другие фирмы.

СРОЧНЫЙ РЕМОНТ БЛОКОВ ПИТАНИЯ

тел. 125 22 12

ВЫ ЗВОНИТЕ – МЫ ПРИЕЗЖАЕМ

- СЖАТЫЕ СРОКИ от 2-х часов до 2-х дней
- На время ремонта ПРЕДОСТАВЛЯЕТСЯ ИСПРАВНЫЙ БЛОК
- ГАРАНТИЯ 3 месяца
- Транспортировка НАШИМ КУРЬЕРОМ

Оплата по безналичному и наличному расчету

А д а - С е р в и с

117292, Москва, а/я 161,
тел. 125 2212

SOFTWARE

SYMANTEC.™

Time Line 4.0
русифицированная версия одной из самых совершенных систем сетевого планирования и управления проектом. Сетевая поддержка.

Zortech C++ 3.0
оптимизирующий транслятор C++, поддерживающий платформы DOS, OS/2, Windows, 16-bit, 32-bit DOS Extenders. В версии Science&Engineering поставляются обширные библиотеки классов с исходными текстами, ориентированные на математические расчеты. Возможна поставка версий для UNIX V и Macintosh System 7.0.

pcANYWHERE IV
популярный коммуникационный пакет, поддерживающий протоколы NetWare/IPX и NetBIOS.

Продукты, созданные Питером Нортоном и его командой:
Norton Commander 3.0, Norton Editor 2.0, Norton Utilities 6.0, Norton AntiVirus 1.5, Norton BackUp 1.5, Norton Desktop for Windows 1.0.

Другие известные продукты для DOS, Windows и Macintosh.

HARDWARE

Плата ELIS-92, имеющая сертификат Центрального телеграфа, позволяющая стыковать PC с телексными и телетайпными линиями. Обеспечивает прием и передачу телеграмм в диалоговом и фоновом режимах. Подробно документированное программное обеспечение.

Плата ввода и обработки ТВ-изображения, обеспечивающая математическую обработку и хранение изображений с видеокамеры и видеомагнитофона в формате PCX. Время ввода кадра от 20мс до 1.5с. Число градаций яркости – 64.

ОФИЦИАЛЬНЫЙ ДИЛЕР

121205, Москва, Новый Арбат, д.56,
офис 2916
Тел.: 290-86-30, 188-33-10, 556-71-39
Факс: 928-07-40
Сетевой адрес
RELCOM:yuri@jet.msk.su

Анонс

Монитор" - версия 5.92

С.Крапивенко. Clipper или Paradox: попытка аргументированного выбора часть 2.
Сравниваются представление данных и техника работы с таблицами.

Л.Ливеровский. Что такое Clarion?
Рассказ об интегрированной среде программирования Clarion.

М.Черкашин. Вычисляем по формуле...
Вторая статья в цикле, посвященном конструированию компиляторов.

И.Василенко, Е.Гулаков. Защищайтесь, сударь! Часть 2
Технология создания систем защиты от копирования.

Р.Глуховский. Работа с DXF файлами в Автокаде
Программирование в Автокаде с использованием формата файла обмена рисунками DXF.

А.Поликарпов. EGA и VGA в текстовых режимах
Программирование адаптеров EGA и VGA.

Д.Гришечкин, В.Оленев, А.Тихонов. Гипертекстовые системы
Технология создания гипертекстовых систем

Д.Солдатенков. Спрайты. Путь от графического редактора до игрушки Часть 1
Вторая статья о том, как программировать компьютерные игры.

Письмо в редакцию

С. Курковский

После того как я прочитал в журнале "Монитор" (№ 1 за 1991 год) этюд для программистов и изложенную в нем задачу о новобранцах, у меня появились мысли, о которых, кажется, было бы интересно рассказать.

Конечно же, доказательство завершимости программы с помощью метода математической индукции можно провести, но оно было бы громоздким, и поэтому приведенное автором этюда доказательство является гораздо более подходящим.

Существует принципиально иной метод общего решения поставленной задачи, который сочетает в себе, с одной стороны доказательство завершимости программы, а с другой является наглядным решением вопроса о конечной позиции шеренги и времени окончания поворотов. Идея решения подобных задач приведена в работе Н.Б. Васильева и др. "Заочные математические олимпиады" (изд. "Наука", 1987).

Возьмем листок в клетку и развернем его так, чтобы линии были направлены под углом 45 градусов к горизонтальной прямой. Каждому солдату в шеренге поставим в соответствие наклонный отрезок единичной длины следующим образом: если новобранец смотрит вправо, то отрезок будет направлен снизу вверх, а если влево - наоборот (рис. 1а). Для всей шеренги составим ломаную, совмещая концы отрезков, соответствующих каждому следующему солдату (рис. 1б). Таким образом, мы получили ломаную AC (рис. 2). Рассмотрим, как отобразятся на ломаной AC изменения, которые будут происходить в строю каждую секунду. Паре солдат, смотрящих в лицо друг другу, будет соответствовать верхний излом ломаной. Через секунду эти два солдата развернутся и верхний излом ломаной превратится в нижний. Очевидно высота самого высокого верхнего излома с каждой секундой уменьшается. И так будет продолжаться до тех пор, пока не останется ни одного верхнего излома - ни одной пары солдат, смотрящих в лицо друг другу. Таким образом, первоначальная ломаная трансформируется в угол ABC прямоугольника ABCD, который по введенному нами условию дальше трансформироваться не будет - движение солдат прекратится. Вообще говоря, зная первоначальное состояние строя и построив соответствующую ломаную с концами в некоторых точках A и C, мы сразу, не проводя моделирования, предложенного автором этюда в своей программе, сможем указать

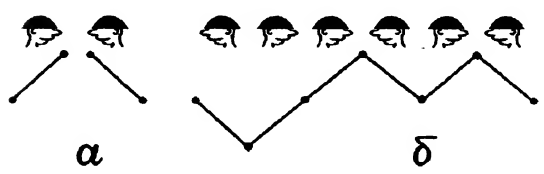


РИС. 1

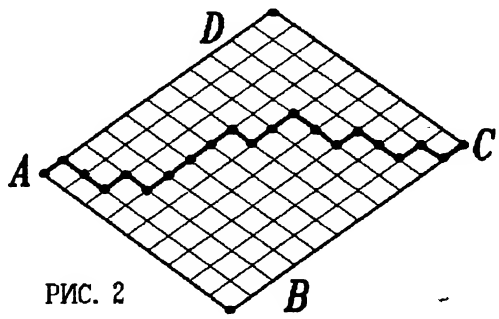


РИС. 2

конечное положение солдат - ему будет соответствовать нижний угол прямоугольника с диагональю AC.

Итак, завершимость программы доказана. Что же касается наибольшего времени, необходимого для того, чтобы строй занял устойчивое положение, то очевидно, что оно потребуется для строя, которому соответствует ломаная ADC, изображенная на рис. 2. Действительно, если в шеренге N солдат и соответственно длина ломаной равна N, то для трансформации ломаной ADC в ломаную ABC потребуется N-1 шаг, а соответствующему строю - N-1 секунда. Для ломаной, лежащей ниже положения ADC (например, для ломаной, приведенной на рис. 2), повороты завершатся за время, меньшее, чем N-1 секунда. ■

Уважаемый читатель!

Всеобщая компьютеризация родных просторов и отечественных весей привела к небывалому духовному подъему советского человека - вечного строителя светлого будущего. Теперь, после свержения жупелов неофеодализма, вся страна с радостью и небывалым энтузиазмом вглядывается в надвигающийся паралич мировой экономики. "Им нас не прокормить!" - радостно восклицают пессимисты. И это, товарищи, правильно. На этой благодатной почве пышным цветом расцветают махровые представители попсово-митьковского толка прозападной ориентации.

Не проходите мимо! Обратите внимание на глубину экспрессии, которой пронизаны эти незамысловатые, на первый взгляд, сюжеты! Сколько в них света и внутренней красоты! А любимая присказка товарища Комманда Кома, одного из главных персонажей цикла: "Дык, оболочки дешевые!" Какое мастерство владения родной речью, какая поразительная степень органичного слияния фольклора, местами сказочного, с сугубо технической информацией.

Именно поэтому редакция приветствует появление нового жанра в отечественной прозе и, желая прослыть в кругах ценителей такого (иногда говорят, английского, хотя это и не совсем верно) юмора, предоставляет свои страницы произведению "Притчи о компьютерных жителях".

В то же время, сознавая, что шедевры появляются редко, просим не присылать нам дешевые подделки. Кроме того, поскольку наш журнал посвящен компьютерной тематике, притчи о пещерных, сельских, таежных и прочих жителях также приниматься не будут!

Притчи о компьютерных жителях

А. Голубев

Когда Диггер сломал себе ноги, он стал передвигаться на колесиках.

* * *

- Поправился товарищ Комманд Ком, - произнес Диггер, рассматривая проходящего мимо товарища Кома. - На 2,7 кило, - определил командир Нортон. - Так какую песню мы сегодня будем петь, Нортон? - спросил Диггер. - Дык, - отвечал командир Нортон, - Янки Дудль.

* * *

Товарищ Комманд Ком жил на системной дискете и вел беспорядочную жизнь, из-за чего страдал. Заразившись, он шел к доктору Нортону. Посадил доктор Нортон товарища Кома в высокое кресло, взял его за нижнюю челюсть: "А теперь, - говорит, - скажи: "format a-a-a-a:"."

* * *

-Как ты думаешь, Нортон, кем мы будем в следующей жизни? - так спрашивал Диггер командира Нортона - Дык, - ответил командир Нортон, нажимая на Контрол-Альт-Дел.

* * *

- Не помните ли Вы Первой Версии Доса, товарищ Ком? - спрашивал Диггер товарища Комманда Кома. - Нет, - плакал тот, - от нее только песня осталась, - и пел, - Ос, Дос, Вер. Дос, а бабушка здорова...

* * *

Сидят как-то Диггер, командир Нортон и товарищ Комманд Ком, а мимо идет отец Вындоуз. "Сидите, - говорит. - Ну, ну. Скоро все подо мной сидеть будете". Никто ему так ничего и не ответил. Один товарищ Комманд Ком сплюнул и грязно выругался. "Проходи, проходи, - сказал, - оболочка дешевая".

* * *

Спросил как-то Диггер товарища Комманда Кома: "За что же Вы, товарищ Комманд Ком, отца Вындоуза так не любите?" "Мышами от него воняет, - быстро ответил тот, но, подумав, добавил, - от оболочки от дешевой".

* * *

"Господа, - сообщил командир Нортон Диггеру и товарищу Комманду Кому, - вчера произошла Великая Русификация. Велено всем присвоить русские имена". Стали они думать. Ну, командира Нортона сразу назвали поручиком Северным, Диггера так Диггером и оставили. А вот товарищу Комманду Кому никакое русское имя не нравилось. Хотели было его Иваном Комовым назвать, да разозлился он: "Оболочки, - кричит, дешевые! Мышей развели".

* * *

- Куда деньги дел? - грубо спросил командир Нортон Диггера. - Дык, съел.

* * *

Весь день товарищ Комманд Ком ходил просветленный. А встретив грустного отца Вындоуза, даже поинтересовался участливо: - Или случилось что, батюшка? - Все иконы украли, - сказал отец Вындоуз и заплакал. - Ах они нехристи, - радостно сочувствовал товарищ Комманд Ком, - ах они оболочки дешевые! И усы его весело шевелились.

* * *

Только собрались Диггер, товарищ Комманд Ком и командир Нортон пивца попить, как в окно к ним постучали. "There ist't enough room!" - закричал командир Нортон, увидев в окне ненавистный куколь отца Вындоуза.

* * *

(Из жизнеописания великого путешественника товарища Комманда Кома).

Поехал как-то товарищ Комманд Ком в Африку, а там его укусила муха Формат Це-Це.

* * *

-Правда, что Вентура отдалась отцу Вындоузу? - спрашивал командир Нортон у товарища Комманда Кома. - Дык, ответил тот, - Паблишер она Паблишер и есть.

* * *

Из жизни кремниевых токсикоманов.

- Худо, Нортон? - Тускняк, товарищ Комманд Ком. - Пойдем к Диггеру. У него хоть колеса есть.

* * *

Пошел как-то Диггер золото воровать, да засыпался.

* * *

Когда командир Нортон оставался один, он заболевал звездной болезнью. ■

ОЧЕНЬ СВОЕВРЕМЕННАЯ КНИГА

ОБШИРНЫЙ **КАТАЛОГ** ЗАПАДНЫХ ФИРМ

Электроника для Офисов, Банков и Торговли
Бытовая Электроника, Компьютеры и Периферия
Разнообразное Программное Обеспечение

Более 6000 фирм
Адреса, Телефоны
Предлагаемые Товары

Стоимость каталога - 985 рублей + 28%
Бланки заказов и копии платежных поручений
просьба высылать по адресу:

Москва, 117419,
а/я 765, Каталог

Счет: НТО "Софт-Москва"
р/с 467002
в КБ "Банк Развития XXI век",
к/с 161816 в РКЦ ГУ ЦБ РФ
по г.Москва, МФО 201791

тел. (095) 247 3625
факс (095) 126 3156

БЛАНК ЗАКАЗА

Название организации _____
Адрес для высылки Каталога _____
_____
_____Телефон
_____
Количество Экземпляров _____

Clipper 5

ВЫБОР ПРОФЕССИОНАЛОВ

Став зарегистрированным пользователем, Вы сможете:

- **обладать полностью русифицированным продуктом и документацией (за рубли);**
- **легально разрабатывать и распространять собственные коммерческие продукты на Clipper;**
- **получать бесперебойные консультации по "горячей" телефонной линии (095) 289– 43– 00.**

Легальная копия– входной билет в мировое сообщество Clipper

За информацией обращайтесь в СП "Магнит".
Наш адрес: 127018, Москва, 2– я Ямская, 15.
Телефон: (095) 289– 44– 77 Факс: (095) 289– 34– 69

Nantucket®